Escola Politècnica Superior de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TREBALL FINAL DE CARRERA

TÍTOL: Avaluació d'una xarxa ad-hoc real amb protocol d'encaminament AODV

AUTOR: Xavi Mantecón Ferrer

DIRECTOR: Carles Gómez Montenegro

CO-DIRECTOR: Marisa Catalán Cid

DATA: 25 de febrer de 2005

Títol: Avaluació d'una xarxa ad-hoc real amb protocol d'encaminament AODV

Autor: Xavi Mantecón Ferrer

Titulació: Enginyeria Tècnica de Telecomunicacions

Especialitat: Telemàtica

Pla: 2000

Director: Carles Gómez Montenegro

Departament: Enginyeria Telemàtica

Vist,

Director del TFC

Registre:

Títol: Avaluació d'una xarxa ad-hoc real amb protocol d'encaminament AODV

Autor: Xavi Mantecón Ferrer

Director: Carles Gómez Montenegro

Data: 25 de febrer de 2005

Resum

Avui en dia, Internet és eina diària per a gairebé tothom. La tecnologia ens proporciona dispositius portàtils de mida cada vegada més reduïda, però amb una capacitat sorprenent de comunicació i una suficient autonomia. Això ha augmentat la demanda de l'ús de xarxes ad-hoc (MANETs) com xarxes d'accés a Internet i xarxes aïllades. L'encaminament és un dels problemes significatius de les MANETs. També ho és el transport. L'AODV (*Ad-Hoc On-demand Distance Vector*) és un dels protocols d'encaminament per aquestes xarxes més rellevants en l'actualitat. ATCP és una millora del TCP per a aquests escenaris. Aquest document estudia i quantifica l'efecte que el protocol AODV causa en una xarxa ad-hoc real, caracteritzant diferents paràmetres de la xarxa en funció de la configuració de l'AODV. Aquest són l'ample de banda, la latència, el consum d'energia i, sobretot, la capacitat de reacció enfront canvis de topologia de la xarxa ad-hoc.

Paraules clau: MANET, AODV, ATCP, protocol encaminament, protocol transport, mesures xarxa reals.

Internet is a daily tool for almost everyone. Technology provides us smaller portable devices with astonishing communication capabilities and enough battery lifetime. The situation has increased the demand of usage of mobile ad-hoc networks (MANETs) as access networks to Internet, as well as isolated networks. Routing is one of the most significant problems of the MANET's networks. Transport is too. AODV (Ad-Hoc On-demand Distance Vector) is one of the most relevant current routing protocols for ad-hoc networks. TCP is a modification of TCP adapted to MANETs. This document studies and quantifies the effect that AODV protocol causes in a real ad-hoc network performance, characterizing the diferent parameters of the network depending of the AODV configuration. The target parameters are bandwidth, latency, energy cost and reactivity after topology changes.

Key words: MANET, AODV, ATCP, routing protocol, transport protocol, real network measurements.

AGRAIMENTS

Vull dedicar aquest treball, de tot cor, a totes les persones que m'han ajudat a fer-lo possible, principalment als meus directors, Carles Gómez i Marisa Catalán, i també al David Garcia per la seva inspiració, i a Rafael Vidal, per la seva benevolència en el préstec de material del laboratori. Per descomptat, aquest treball no hauria estat possible sense la implementació d'AODV realitzada per l'Erik Nordström. Li agraeixo la seva col·laboració en aquest treball, així com també a en Harkirat Singh, un dels coautors de l'ATCP.

No m'oblido, ni molt menys, de la família, parella, amics i companys de classe, ni tampoc dels professors que he tingut al llarg de la meva vida. Sense tots vosaltres ben segur que tot això no hauria estat possible. Ja sabeu qui sou, i per mi hi sou tots. Moltes gràcies.

INDEX

INTRODUCCIÓ

Ens trobem a l'era de la informació. Necessitem estar comunicats en qualsevol lloc i en qualsevol moment, i la tecnologia ens proporciona dispositius cada vegada més petits, amb majors possibilitats de connexió i amb una major autonomia. Això ha motivat l'aparició de les xarxes sense fils. Amb aquestes xarxes, existeix la possibilitat d'accedir a Internet o compartir la informació dels nostres dispositius sense necessitat d'utilitzar cables. Aquestes xarxes, però, suposen un nou repte per als enginyers, ja que disposen d'unes característiques que les fan diferents de les xarxes tradicionals. Les xarxes sense fils són més insegures que les xarxes amb cable, i el medi de transmissió és més hostil. A més, normalment ens trobem amb uns dispositius amb una capacitat de procés reduïda i amb una vida de bateria limitada.

En aquest treball ens centrarem principalment en les xarxes ad-hoc, també conegudes com MANETs (*Mobile* Ad-Hoc *Networks*). Les xarxes ad-hoc són xarxes sense fils que no necessiten d'una infraestructura prèvia. Es pretén que els dispositius es reconeguin els uns els altres i que siguin capaços d'establir comunicacions automàticament. Ens trobem, doncs, amb una topologia el més variable possible, i aquest és possiblement un dels principals motius d'estudi i investigació en aquest camp. Necessitem de nous protocols d'encaminament. N'existeixen principalment de dos tipus, els proactius i els reactius. Amb els protocols proactius, els dispositius disposen d'un mapa actualitzat de la xarxa en qualsevol moment, mentre que amb els reactius, un node ha de fer un descobriment de ruta per arribar al destí. En aquest treball, estudiarem el protocol reactiu AODV (Ad-Hoc *On Demand Distance Vector*), segurament el més estudiat i representatiu d'aquest grup. La inestabilitat del medi de transmissió és un problema general de les xarxes sense fils. En aquestes condicions, els protocols de transport també necessiten ser modificats per poder treballar amb més eficiència. L'ATCP (Ad-Hoc *Transmission Control Protocol*) és una modificació del TCP adaptada al nou medi de transmissió.

En aquest treball es pretén analitzar el rendiment d'una xarxa ad-hoc real. Estudiarem com funcionen els protocols AODV i ATCP, i veurem quin és l'efecte de la configuració de l'AODV en el rendiment de la xarxa ad-hoc.

El document es divideix en sis capítols. En el primer capítol estudiarem les aplicacions principals i els problemes que poden aparèixer en les xarxes sense fils, concretament les xarxes ad-hoc. En el segon capítol mencionarem les característiques principals del protocol d'encaminament AODV, mentre que en el tercer capítol veurem quins són els problemes de TCP, i com l'ATCP podria millorar el seu rendiment. Justificarem l'elecció de les implementacions dels protocols usades en aquest estudi. En el quart capítol detallarem com muntar una xarxa ad-hoc en un escenari real. En el cinquè capítol presentarem els escenaris de proves per analitzar el rendiment d'aquests protocols. Finalment, en el sisè capítol, estudiarem els resultats i n'extraurem conclusions.

Els resultats obtinguts de les proves realitzades en aquest estudi provenen de situacions reals i no d'aproximacions obtingudes amb simulacions, essent aquest l'atractiu principal d'aquest treball.

CAPÍTOL 1 INTRODUCCIÓ A LES XARXES AD-HOC

1.1 Orígens i definició

Les xarxes mòbils ad-hoc[1], també conegudes com *Mobile* Ad-Hoc *Networks* (MANETs [1]), són un conjunt de dos o més nodes capaços d'autoconfigurar-se[2] i de comunicar-se entre ells, sense fils i sense necessitat d'una infraestructura prèvia. Qualsevol d'aquests nodes pot actuar tant com un host com un router, és a dir, pot ser transmissor i/o receptor de la comunicació, o bé limitar-se a reencaminar la informació. Per a que això sigui possible sense la existència d'un node central que faci les funcions de router, com és en el cas del mode infraestructura, han d'existir uns protocols d'encaminament que permetin als nodes disposar, en tot moment o quan és necessari, d'informació sobre com encaminar paquets d'un origen a un destí.

L'origen d'aquestes xarxes és militar. A mitjans dels anys 70, es van començar a investigar les possibilitats de les comunicacions sense fils. Així, els militars utilitzaven unes primitives xarxes sense fils amb les quals podien transmetre informació als seus aliats. A part de la possibilitat de estar comunicat en qualsevol lloc i en qualsevol moment, la característica més innovadora d'aquestes xarxes era que es podia transmetre informació a equips que no estiguessin en el mateix rang de cobertura, utilitzant nodes intermedis com a retransmissors. No existia la necessitat d'haver muntat una infraestructura estacionaria prèvia, que podria ser un fàcil objectiu militar. Així es podria reconfigurar la xarxa en cas d'emergència.

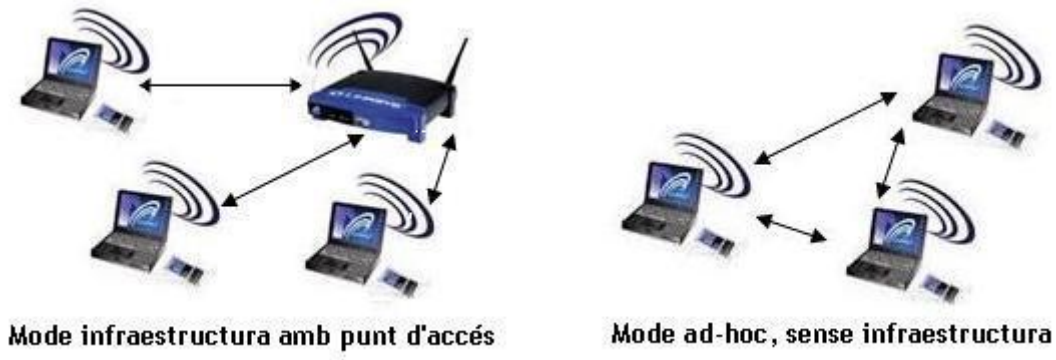


Fig. 1.1 Topologies de xarxa: infraestructura vs ad-hoc.

Avui en dia, Internet és eina diària per a gairebé tothom, i la tecnologia ens proporciona uns dispositius cada vegada més petits, però amb unes capacitats sorprenents de comunicació i una suficient autonomia. Actualment no és

[1] Del llatí. Significa "per a això", amb connotació d'improvisació.
[2] Existeix un grup de treball de l'IETF que investiga aquest camp, però les tasques de desenvolupament i estandardització es troben encara en procés.

estrany accedir a Internet des d'un dispositiu autònom, com un portàtil o una PDA, això sí, gairebé sempre a través dels coneguts punts d'accés, que han d'estar muntats i configurats prèviament. Tot això ha despertat un altre cop la motivació de l'estudi de les possibilitats de les xarxes ad-hoc.

1.2 Característiques i problemes principals de les xarxes ad-hoc

La característica principal de les xarxes ad-hoc és que no necessitem de cap tipus de muntatge o d'infraestructura prèvia. Això permet desplegar una nova xarxa de manera fàcil i ràpida. Tan aviat com els nodes es troben suficientment a prop com per poder comunicar-se, es forma la xarxa sense fils. Els dispositius han de ser capaços de configurar-se per si mateixos, i de trobar la ruta més adequada per establir o reencaminar una comunicació.

Una altra propietat important és que podríem considerar aquestes xarxes com autònomes, ja que normalment només ofereixen connectivitat entre els nodes que la formen, i no cap a l'exterior o Internet. Existeix però una configuració híbrida entre mode infraestructura i ad-hoc. Un dels nodes pot actuar com a *gateway* o porta d'enllaç i proporcionar accés a Internet o altres xarxes a la resta de veïns.

En el món ad-hoc existeixen molts problemes addicionals i diferències respecte les xarxes amb cable convencionals. A continuació en destaquem els més interessants.

1.2.1 Arquitectura de protocols

L'arquitectura de protocols TCP/IP ha demostrat ser una estructura fiable i molt adaptable a qualsevol medi o entorn. Les xarxes ad-hoc no haurien de ser una excepció, i així es mantindria la compatibilitat amb l'Internet existent. Les característiques d'aquestes xarxes, però, degraden a vegades en excés el seu rendiment. En les xarxes ad-hoc la topologia és variable en tot moment, fet que provoca que hi hagi trencaments i/o canvis de ruta. El canal de transmissió té també una taxa d'error més elevada que amb el cablejat tradicional. Tot això provoca que el rendiment d'alguns protocols, com és el cas del protocol de transport TCP, es degradin notablement.

S'ha demostrat que una altra arquitectura de protocols, depenent de les nostres necessitats, donaria uns millors resultats, però seria incompatible amb la majoria de dispositius a dia d'avui, ja que aquests implementen l'arquitectura TCP/IP. És millor , doncs, adaptar només algunes de les capes de l'arquitectura de protocols actual. És el que intenta fer el protocol de transport ATCP, que estudiarem més endavant.

1.2.2 Característiques dels dispositius

Els dispositius que formen aquestes xarxes poden ser del més variat. Podem tenir connectats una estació multiprocessador de treball amb dispositius de menor rendiment, com PDAs. El més normal en aquest tipus de xarxes és que els dispositius tinguin unes característiques limitades, sobretot parlant en termes de capacitat de procés, memòria i autonomia de la bateria. Aquesta última característica és, segurament, un dels elements més apreciats en el món ad-hoc. S'han d'evitar les retransmissions innecessàries i entrar en mode repòs quan sigui possible, considerant el nivell d'enllaç, i s'ha de reduir, en la mida possible, la freqüència de missatges de control en el nivell de xarxa.

1.2.3 Accés al medi

Tots els nodes utilitzen i comparteixen el medi aire per transmetre. Existeixen protocols MAC (*Medium Access Control*), com el 802.11 de la IEEE, que eviten que dos nodes transmetin a la vegada, ja que si això succeeix les dues transmissions col·lisionen i es perd la informació. Gairebé en tots els estudis, s'assumeix que el 802.11 és el nivell d'enllaç utilitzat en la realització de les xarxes ad-hoc, encara que es poden utilitzar altres tecnologies de la IEEE com Bluetooth i ZigBee (veure [2]) Existeixen, però, problemes associats en general a les xarxes sense fils, com són, per exemple, el problema del terminal ocult i del node exposat.

En el cas del terminal ocult, existeixen dos nodes, sense comunicació directa entre ells, que volen transmetre dades a un node intermedi, dins l'abast de transmissió dels dos primers. Imaginem que els dos nodes dels extrems volen transmetre al node intermedi a la vegada. Escolten el medi i esta lliure, ja que no escolten les emissions l'un de l'altre. Tots dos nodes comencen a transmetre i les dades col·lisionen, perdent-se, en el node intermedi. Per solucionar aquest problema, existeixen protocols de *handshake* com el RTS/CTS (*Ready To Send / Clear To Send*), tot i que no és la solució definitiva, ja que també pot succeir que col·lisionin dos missatges CTS i RTS en un mateix node.

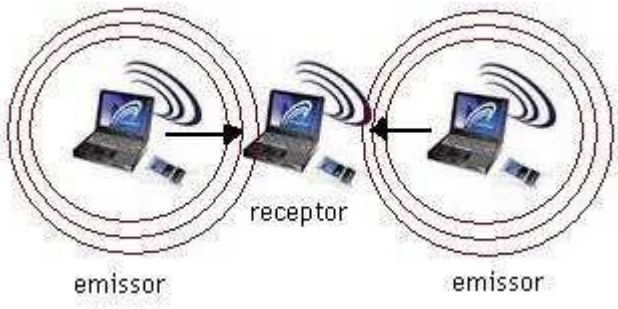


Fig. 1.2 Problema del terminal ocult.

En els cas del terminal exposat, imaginem que un node vol transmetre dades a un segon, i al escoltar el medi hi ha un tercer node transmetent. El primer node cancel·la la transmissió, mentre que la transmissió del tercer node no afecta al segon. En aquest cas, el primer hauria pogut transmetre sense problemes. Es pot entendre millor amb l'ajuda de la figura 1.3.

Fig. 1.3 Problema del terminal exposat.

1.2.4 Adreçament

Les xarxes convencionals disposen de servidors de configuració TCP/IP com DHCP (*Dynamic Host Configuration Protocol*), que són accedits pels nodes quan necessiten modificar o a accedir a alguna de les seves propietats de xarxa. Les xarxes ad-hoc no disposen d'aquest mecanisme centralitzat, fet que ha motivat l'aparició de propostes per a l'autoconfiguració de adreces IP.

Recentment ha aparegut un nou grup de treball de l'IETF (*Internet Engineering Task Force*), el MANET-AUTOCONFIG, que treballa per estandarditzar els estudis realitzats sobre autoconfiguració d'adreces IP.

1.2.5 Topologia i encaminament

En una xarxa amb cables convencional, existeix un dispositiu hardware especialitzat, o un ordinador amb el software i interfícies apropiades, que fa les funcions de *router* o encaminador de la informació. Tots els nodes que volen accedir a Internet o comunicar-se amb altres equips de la xarxa envien per defecte la informació a aquest equip, i és aquest equip l'encarregat de encaminar les dades allà on han d'arribar.

En les xarxes ad-hoc, la ruta és calculada pels propis nodes participants, amb l'ajuda de protocols d'encaminament propis per a aquests tipus d'estructura, ja que a més del problema de l'absència d'infraestructura, ens trobem amb una topologia variable.

1.2.6 Seguretat de la xarxa

Tema molt controvertit, ja que precisament en una xarxa ad-hoc no tenim perquè conèixer ni la identitat ni les intencions dels nostres veïns. A més, és més fàcil atacar la informació que viatja pel medi aire que no pas la que viatja a través d'un cable. La criptografia convencional d'intercanvi de claus i infraestructura pública de claus no funciona en aquest tipus de xarxes, ja que no disposem de la infraestructura prèvia ni d'autoritats de confiança. La solució passa per securitzar els protocols típics d'encaminament d'aquestes xarxes.

1.3 Encaminament

Ja s'ha esmentat que l'encaminament en les xarxes ad-hoc ha de ser necessàriament específic per a aquest tipus de xarxes. Això és a causa del ample de banda disponible, que és més limitat, i de l'augment de la freqüència amb la que es produeixen trencaments i canvis de topologia a la xarxa.

Els protocols d'encaminament són els responsables de determinar com i cada quan els nodes d'una xarxa han d'enviar-se informació per tal de trobar els camins als possibles destins. Els protocols d'encaminament tradicionals de les xarxes amb cables es classifiquen principalment en dues branques ben diferenciades, segons l'algoritme que utilitzen. L'algorisme és l'encarregat de determinar el camí per arribar als destins, amb la informació proporcionada pels protocols d'encaminament. Existeixen principalment els protocols basats en algoritmes de vector distància (p.e: algoritme Bellman-Ford distribuït), i els basats en algoritmes d'estat de l'enllaç (p.e: algoritme Dijkstra).

Usant algoritmes basats en el vector distància, els nodes tenen un coneixement parcial de la xarxa, intercanviant les taules d'encaminament, parcial o íntegrament, amb els seus veïns. En destaquem el protocol RIP. Del segon grup, el protocol més representatiu és l'OSPF. Utilitzant aquest protocol, els nodes obtenen un mapa global de la xarxa intercanviant l'estat dels seus enllaços amb tots els altres nodes de la xarxa.

Són algoritmes que fins ara havien funcionat molt bé, però amb les noves característiques mòbils dels nodes, s'han de trobar nous protocols que s'adaptin millor a escenaris amb una taxa de canvis de topologia força elevada i un ample de banda limitat. Això és el que ha motivat l'estudi de noves solucions per part dels membres de la comunitat científica. Aquest estudis han tornat a divergir en dues branques principals, els protocols que anomenem proactius i reactius. Reben aquesta nomenclatura en funció de la naturalesa proactiva o reactiva que escullen per determinar el camí per on encaminar la informació.

1.3.1 Protocols proactius

Utilitzant protocols proactius, els nodes mantenen una comunicació periòdica intercanviant missatges d'informació entre ells. Així, poden conèixer la

topologia de la xarxa en qualsevol moment, encara que no hi hagi transferència d'informació sota demanda, a costa de malgastar recursos com la bateria quan potser no és necessari. Però gràcies als missatges periòdics *broadcast*, tots els nodes disposen d'una taula d'encaminament consistent i actualitzada periòdicament. També són coneguts com protocols *table-driven*. Els exemples més representatius són l'OLSR (Optimized *Link State Routing*) i el TBRPF (*Topology Dissemination Based on Reverse-Path Forwarding)*. Mencionem també el DSDV (*Destination Sequence Distance Vector*), precursor dels anteriors però ja obsolet (veure [3]).

1.3.2 Protocols reactius

Utilitzant protocols reactius, els nodes només busquen o mantenen informació sobre encaminament quan inicien o cursen una transferència d'informació, respectivament. Per això són també coneguts com a protocols sota demanda o *on-demand driven*. D'aquesta manera s'optimitzen els recursos, ja que no hi ha comunicació si no hi ha una acció sota demanda. En canvi, existeix una certa latència a l'hora d'iniciar les comunicacions, ja que els nodes han de descobrir les rutes als possibles destins. Els exemples més representatius que trobem d'aquest tipus de protocol són l'AODV (*Ad-hoc On-demand Distance Vector Routing*), el DSR (*Dynamic Source Routing*) i el TORA (*Temporally Ordered Routing Algorithm*) (veure [4]).

1.3.3 Conclusions

Com veiem, cada família de protocols d'encaminament ad-hoc té els seus avantatges i desavantatges. Existeixen aleshores solucions mixtes, que intenten implementar el millor de les dues famílies. Són els protocols híbrids, i el més rellevant és el ZRP (*Zone Routing Protocol* [5]).

El protocol ZRP defineix zones dins la xarxa; a l'interior de les zones s'actua de manera proactiva, mentre que entre les zones l'encaminament és reactiu. Una optimització de ZRP és SHARP (*Sharp Hybrid Adaptive Routing Protocol*), que permet que les zones d'encaminament proactiu augmentin o disminueixin sensiblement a la mobilitat dels nodes.

Per aconseguir uns resultats òptims, no podem fer un altra cosa que basar la nostra elecció en funció dels patrons de mobilitat i de tràfic de la xarxa que volem realitzar, tenint en compte també les característiques dels nodes que han de formar aquesta xarxa.

En xarxes on els nodes s'alimenten d'una bateria i el patró del tràfic és a ràfegues molt distanciades entre elles, existint llargs períodes d'inactivitat, potser la millor elecció és un protocol reactiu, ja que quan no hi hagi comunicacions actives no s'enviaran missatges a la xarxa, allargant així la vida dels dispositius. En qualsevol cas, no existeix una implementació que sigui òptima en tots els escenaris possibles.

1.4 Aplicacions

Són moltes les aplicacions noves que ens podem plantejar. A continuació llistarem les principals aplicacions de les xarxes ad-hoc pures:

- Xarxes espontànies: companys de feina, participants en conferències o estudiants poden compartir els seus documents, materials de presentació o qualsevol altre tipus d'informació en format electrònic. Aquestes són les conegudes com PAN (Personal Area Networks).

- Àrees de desastre: en el cas de que ocorregués un desastre, com pot ser un terratrèmol o un incendi, es poden restablir les comunicacions si la infraestructura de comunicacions per cable ha resultat danyada.

- Aplicacions militars: comunicacions entre tancs, avions, vaixells en moviment (no oblidem l'origen de les xarxes ad-hoc).

- Construcció de xarxes sense fils: en aquells llocs on posar cable és molt difícil o gairebé impossible per qüestions arquitectòniques, és possible desplegar una xarxa de nodes que permetin configurar un camí visible entre els diferents punts.

- Xarxes de sensors: aquestes xarxes poden ser de gran utilitat, a part de la varietat d'aplicacions que podem considerar dins aquest camp. Per exemple, podem desplegar xarxes de petitíssims sensors sense fils i independents que analitzen dades del medi on es troben. Aquests dispositius envien informació a un centre de control que analitzarà totes aquestes dades, extraient-ne les conclusions necessàries. Sensors de temperatura, foc o radiacions... Molta varietat, que amb una mica de imaginació, obre un ventall infinit de noves possibilitats.

Considerant també, tot i que no són motiu d'estudi en aquest treball, les xarxes ad-hoc híbrides, trobem altres aplicacions. Aquestes xarxes permeten l'accés a altres xarxes, com Internet. Podem doncs configurar la interconnexió de xarxes a través d'un PC, que pot actuar com *gateway* de la xarxa ad-hoc cap a Internet o una altra xarxa. Això pot ser aplicable a una oficina o bé a una llar, amb un PC recollint les dades d'Internet i distribuint-les als electrodomèstics que formarien una xarxa ad-hoc.

La tecnologia és la causa de que tot això sigui possible. Integrar una capacitat de comunicació fins ara desconeguda en dispositius més petits que una moneda i amb autonomia pròpia, a un preu realment assequible, és obra i gràcia dels esforços i diners que s'estan invertint en el camp de les comunicacions sense fils. L'estandardització dels protocols d'encaminament en xarxes ad-hoc permetrà als fabricants explotar aquesta tecnologia, apropant a la vida diària noves aplicacions fins ara inimaginables.

CAPÍTOL 2 EL PROTOCOL D'ENCAMINAMENT AODV

En aquest projecte s'avalua una xarxa ad-hoc on els nodes empren el protocol d'encaminament AODV. En el tram inicial d'aquest capítol, estudiarem el funcionament del protocol d'encaminament reactiu AODV. Aquest és un dels protocols d'encaminament més representatius del món ad-hoc. Els nodes busquen el camí cap a un destí mitjançant un procediment de demanda de ruta. Aquest descobriment afegeix una certa latència inicial a l'hora d'establir la comunicació, però utilitzant AODV, s'introdueix poc tràfic de control en unes xarxes on l'ample de banda és un recurs escàs. En la darrera part del capítol, veurem quines són les implementacions disponibles del protocol, i justificarem perquè hem escollit la implementació de la Universitat de Uppsala, realitzada per Erik Nordström, per dur a terme les proves del nostre estudi.

2.1 Visió general

AODV (*Ad-Hoc On-Demand Distance Vector*) és un dels protocols reactius creats especialment per a les xarxes ad-hoc. Quan cap dels nodes envia informació, no s'envia cap missatge de control AODV. Si un dels nodes necessita establir una comunicació, inicia un procés de descobriment de ruta. Els nodes veïns envien missatges de control[3] per la xarxa per tal de trobar la ruta necessària, i un cop trobada, la mantenen a les seves taules mentre la comunicació roman activa. Mentre perdura aquesta comunicació, només els nodes que participen en la ruta activa es comuniquen mitjançant missatges de control. Un cop la comunicació finalitza, els nodes participants esborren de les taules d'encaminament les entrades associades a la ruta inactiva. Si no participen en cap més comunicació, no envien més tràfic de control fins que no torna a haver-hi un nou descobriment.

Després de 13 *drafts*, l'especificació d'aquest protocol ha estat publicada el juliol de 2003, en el RFC 3561, encara en estat experimental. Els autors d'aquest document són en Charles E. Perkins, investigador de Nokia, E. Belding-Royer, de la Universitat de California i Sajal K. Das, de la Universitat de Cincinnati (veure [6]).

Els orígens de AODV provenen del DSDV (*Destination-Sequenced Distance-Vector*). El protocol DSDV es caracteritza per inundar la xarxa amb missatges *broadcast* de control cada vegada que hi ha un canvi en la topologia. Cada vegada que dos nodes entren dins del rang d'emissió l'un de l'altre, esdevenen veïns, canviant la topologia de la xarxa. Aquest fet fa que la xarxa sencera s'inundi amb missatges de control *broadcast* anunciant el canvi. El mateix passa quan dos nodes es separen.

Amb AODV, quan hi ha un canvi a la topologia de la xarxa, no s'anuncia cap canvi si el node que entra o surt de la xarxa no participa en alguna de les rutes

[3] Aquest missatges viatgen en paquets UDP pel port 654.

actives. A diferència del que succeeix amb el protocol DSDV , si un enllaç es trenca, només s'informa als nodes que l'estaven utilitzant, i no a la resta de la xarxa. Una altra característica de l'AODV és que no modifica o afegeix cap *overhead* als paquets que transporten les dades. Quan una ruta no s'utilitza, s'esborra, i així ens evitem un manteniment innecessari. És, doncs, un protocol minimalista, pensat en principi per l'encaminament en xarxes sense fils, però res no el priva del seu bon funcionament en les xarxes amb cable convencionals. AODV també ofereix encaminament *multicast*.

S'especifiquen tres tipus de missatge AODV. Els *Route Request* (RREQ) són utilitzats quan un node vol descobrir la ruta cap a un destí. Si un node sap com arribar a aquest, o bé és el destí mateix, respon amb un *Route Reply* (RREP). Els *Route Error* (RERR) són enviats quan hi ha trencaments, per advertir els enllaços trencats als nodes que els utilitzaven, procedint així aquests a fer un nou descobriment si correspon. En l'annex A es detalla el format d'aquests missatges.

2.1.1 Números de seqüència

AODV és capaç de mantenir les rutes lliures de bucles, i això ho aconsegueix amb l'ús de números de seqüència. Cada node manté el seu propi número de seqüència. Per a que tot funcioni correctament, els nodes han de tenir actualitzat, per a un destí determinat, el camp *Destination Sequence Number* (DSN) a la entrada corresponent de la taula d'encaminament. El DSN és el número de seqüència associat al destí, i és actualitzat cada vegada que un node rep informació relacionada amb aquest, a través dels missatges AODV. Comparant el DSN dels missatges AODV rebuts amb el valor existent a la taula d'encaminament, s'assegura que un node sempre utilitzarà la ruta més actualitzada.

Els nodes actualitzen el seu número de seqüència en els següents casos:

- L'incrementen en una unitat, cada vegada que generen un RREQ.
- Abans que un node destí generi un RREP en resposta a un RREQ, té que actualitzar-lo al valor més gran entre el seu propi número de seqüència i el DSN del RREQ. Pot succeir que el node que genera el RREQ tingui un DSN associat al destí més gran que el propi número de seqüència del destí, si ha existit un procés RERR.

Els nodes actualitzen els números de seqüència associats a un destí en els següents casos:

- Reben nova informació sobre el número de seqüència d'un destí determinat mitjançant els missatges AODV que viatgen per la xarxa.
- Si s'adverteix el trencament d'una ruta cap a un destí mitjançant l'absència de missatges de *hello*, o bé al rebre un RERR, incrementen en una unitat el DSN associat a aquest destí.
- Si intenten reparar la ruta cap a un destí, incrementen en una unitat el DSN associat a aquest.
- Si la ruta cap al destí expira, el DSN associat s'inicialitza a 0.

2.1.2 Entrades de les taules d'encaminament

Quan un node escolta un missatge de control AODV d'algun dels seus veïns, crea o actualitza una ruta cap a un destí. El primer que fa és consultar la seva taula d'encaminament per comprovar si existeix alguna entrada associada als destins, ja que pot ser que aquests missatges continguin informació actualitzada sobre algun dels destins existents a la seva taula d'encaminament. Si no hi ha entrada a la taula es crea una entrada nova, i si existeix, serà actualitzada si correspon. Aquesta només s'actualitza si:

- el número de seqüència del missatge és més gran que el de l'entrada de la taula d'encaminament del node.
- els números de seqüència són iguals, però el nombre de salts fins al destí (*hop count*) existent al missatge, incrementat en una unitat, és menor que el valor emmagatzemat a la taula.
- el número de seqüència per al destí és desconegut a la taula d'encaminament.

El temps de vida d'aquesta ruta s'extrau dels mateixos paquets de control. Si no existeix aquesta informació, s'inicialitza el temps de vida a un valor ACTIVE_ROUTE_TIMEOUT[4]. El temps de vida d'una ruta és actualitzat un altre cop a aquest valor amb cada paquet de dades encaminat, mentre la ruta resta activa. Quan la ruta deixa de ser utilitzada, expira el seu temps de vida d'una ruta, i és aleshores marcada com invalida. Un cop invalidada, la ruta no pot ser utilitzada, però es guarda la informació coneguda. Aquesta no pot ser esborrada fins passat un temps definit a DELETE_PERIOD.

A l'entrada corresponent a una ruta cap a un destí, es guarden les següents dades:

- la direcció IP del node de destí.
- la direcció IP del següent node on s'ha de reenviar la informació dirigida a aquest destí. Aquest node també es coneix com *next-hop.*
- la direcció IP del *next-hop* en la direcció inversa cap a l'origen, si aquest existeix. Aquest node es coneix com el precursor en la ruta cap al destí. Són aquells nodes veïns dels que s'ha rebut un RREQ i als que se li ha contestat o reenviat un RREP. Els nodes precursors seran notificats amb un RERR si el node detecta un trencament de l'enllaç a la ruta, per tal que l'origen pugui dur a terme un altre descobriment.
- el DSN associat a cadascun d'ells.
- la interfície de xarxa associada, el temps de vida i el nombre de salts per arribar al destí.

Es guarda la informació de la ruta directa, cap el destí, i també informació que serveix per configurar el camí de tornada cap a l'origen, la ruta inversa. Amb AODV, es sobreentén que les comunicacions són bidireccionals i que s'utilitzen enllaços simètrics. Aquesta informació proporciona a AODV la capacitat de mantenir els enllaços de les rutes actives i gestionar els trencaments si aquests es produeixen, com veurem més endavant. Existeixen *flags* que indiquen si la ruta està activa, invalidada o si s'està reparant.

[4] ACTIVE_ROUTE_TIMEOUT = 3000 ms (per defecte)

2.2 Funcionament

2.2.1 Descobriment de ruta

Quan un node vol enviar un paquet al destí, el primer que fa es buscar a la seva taula d'encaminament si hi ha alguna ruta associada. Si aquesta ruta existeix, i és vàlida i no ha expirat, s'enviarà el paquet al *next-hop*. Si la ruta no existeix, o no és vàlida, s'ha d'iniciar el procés de descobriment.

En aquest procés, l'origen crea un missatge RREQ. Aquest paquet conté la direcció IP i el número de seqüència actual del node origen (OSN), així com la IP de destí i l'últim DSN conegut associat a aquest. Si no es coneix informació prèvia invalidada, el node ha d'activar el flag *Unknow Sequence Number* del paquet RREQ. El missatge també conté un identificador propi (RREQ ID), que s'incrementa cada vegada que el node crea un nou paquet RREQ per a un destí. La parella que identifica a un missatge RREQ com a únic és la IP d'origen i el RREQ ID. Una vegada el paquet s'ha creat, s'inicialitza el camp de *hop count* a 0 i s'envia a la xarxa.

Per controlar la disseminació dels missatges per la xarxa, es pot utilitzar la tècnica *Expanding Ring Search*. Amb aquesta tècnica, es limita l'abast dels missatges *broadcast* RREQ. Si no fem servir una disseminació controlada, cada vegada que un node inicia un descobriment de ruta, els missatges RREQ arriben a tota la xarxa. Si la xarxa és petita, l'impacte d'aquest mètode de descobriment és mínim, però podem tenir problemes quan la xarxa rebassa un cert nombre de nodes. Podeu trobar més informació a l'annex A.

Mentre es busca la ruta, les dades a transmetre han de ser guardades en un *buffer* del tipus FIFO (*First In First Out*). Si ja s'han realitzat tots els reintents de descobriment de ruta i no s'ha trobat cap camí al destí, es notifica a l'aplicació amb un missatge del tipus *Destination Unreachable* i s'eliminen les dades existents al *buffer*.

Quan s'ha utilitzat una ruta però ja no es fa servir, aquesta es guarda durant un temps a les taules, marcada com invàlida. Si més tard s'ha d'iniciar un altre descobriment a aquest mateix destí, el primer RREQ tindrà un *Time to Live*[5] (TTL) amb un valor igual al nombre de salts emmagatzemats a la ruta invalidada, incrementat en el valor definit a TTL_INCREMENT. Així s'espera trobar al destí ràpidament a prop de la zona on va ser vist per últim cop.

El node que rep el RREQ analitza els dos camps principals, la IP d'origen i el RREQ ID. Els nodes guarden informació sobre els RREQ rebuts en un registre durant un interval de temps PATH_DISCOVERY_TIME[6]. Si ja ha rebut un RREQ amb la mateixa IP d'origen i RREQ ID durant aquest temps, el node descarta el paquet. Si no és així, processa el paquet i en guarda la informació, sempre que sigui més nova que la que ja té. Aquesta informació li serveix per configurar la ruta inversa, és a dir, el camí de tornada cap a l'emissor del

[5] Nombre màxim de salts que pot fer el paquet abans de ser eliminat.
[6] PATH_DISCOVERY_TIME = 2 * NET_TRAVERSAL_TIME (per defecte)

RREQ. La entrada de la ruta inversa a la taula d'encaminament consisteix en la IP del node creador del RREQ, el seu número de seqüència i el nombre de salts que hi ha per arribar fins a ell. El *next-hop* és el node del que ha rebut el RREQ. Aquesta entrada de ruta inversa té un valor de vida específic. Si no és utilitzada durant aquest temps, és esborrada de les taules, per evitar guardar informació innecessària.

2.2.2 Establiment de la ruta

Una vegada processat el RREQ, per respondre amb un RREP, el node ha de tenir una entrada vàlida cap al destí, o ser el propi destí, naturalment, actuant de manera diferent en tots dos casos. En el cas que contesti un node que no sigui el destí, si l'entrada cap a aquest no ha estat invalidada, ha de tenir un número de seqüència associat amb un valor igual o més gran al indicat en el RREQ. És d'aquesta manera com s'eviten els bucles, assegurant amb els números de seqüència que la ruta retornada mai no és antiga. A continuació examinem en detall què succeeix en les diferents situacions.

Si no es coneix la ruta cap al destí indicat en el RREQ, no és vàlida, o el DSN del RREQ és més nou que el existent a la taula del node receptor, es torna a fer un *broadcast* del RREQ, si així ho permet el valor TTL de la capçalera IP del missatge, guardant-ne abans la informació per configurar la ruta inversa. S'incrementa en una unitat el *hop count* i es decrementa en una unitat el TTL del missatge, i al camp DSN se li assigna el màxim entre el valor d'aquest camp en el RREQ i el propi valor a la taula d'encaminament del node que reenvia. Aquest node però, no modifica el DSN de la seva taula.

Si el node que rep el RREQ és el destí, guarda la informació per a la ruta inversa en la seva taula d'encaminament. Aleshores compara el DSN del missatge amb el seu propi, i actualitza aquest últim al valor més gran de tots dos. Aleshores imprimeix aquest valor i el temps de vida (MY_ROUTE_TIMEOUT[7]) de la nova ruta en un RREP, assignant-li el *hop count* a 0. Aquest missatge AODV és dirigit al node del que ha rebut el RREQ.

El node que rep el RREP incrementa en una unitat el *hop count* d'aquest missatge. Es crea l'entrada per la ruta directa amb la informació extreta del RREP. El *next-hop* és el node del que ha rebut el missatge RREP. Si el node receptor del RREP és l'origen, s'inicia la transmissió de dades. Si no és així, es reenvia el RREP al *next-hop* de la ruta inversa. L'entrada associada a la ruta inversa, creada durant el reenviament del RREQ, és actualitzada en reenviar el RREP cap a l'origen. Així s'estableix el camí bidireccional.

Si el node que rep el RREQ és un node intermedi entre l'origen i el destí, que té una ruta vàlida cap a aquest últim, envia el RREP cap al node del que ha rebut el RREQ (sempre que el seu DSN associat al destí sigui més gran que el indicat al RREP). Aquest RREP conté el nombre de salts que el separen a ell del destí, el DSN conegut associat a aquest, i el seu temps de validesa de la ruta. El node del que ha rebut el RREQ és el precursor en la seva ruta cap al

[7] MY_ROUTE_TIMEOUT = 2 * ACTIVE_ROUTE_TIMEOUT (per defecte)

destí. En canvi, el *next-hop* cap al destí serà el node precursor en la ruta inversa. En aquest cas, el destí no sabrà de l'existència de l'origen, si no ha utilitzat una ruta cap a l'origen recentment, ja que el node intermedi no reenvia el RREQ als altres nodes. Haurà de fer un descobriment ell mateix si la comunicació és bidireccional. Per evitar això, es pot activar el *flag Gratuitous RREP* en el RREQ inicial. Més informació a l'annex A.

2.2.3 Manteniment i gestió de les rutes

2.2.3.1 Manteniment local: missatges de *hello*

La informació de veïnat s'obté dels missatges *broadcast* enviats als veïns en el radi d'un solsalt. Aquest són els missatges de *hello*, un tipus especial de RREP. Cada vegada que un node rep un d'aquest missatges, actualitza el temps de vida per la ruta cap a aquest node a la seva taula d'encaminament. Si el node no té cap entrada a la taula pel veí que ha enviat el *hello*, n'hi insereix una de nova.Els missatges de *hello* només són enviats pels nodes que participen en una ruta activa, de manera periòdica a intervals HELLO_INTERVAL[8]. Amb aquest missatges, un node pot advertir a la resta de nodes veïns que encara es troba en el veïnat. Podem considerar aquest missatges de *hello* com a RREP no sol·licitats.

La direcció IP de destí és la IP del node, el DSN el seu número de seqüència i el *hop count* té un valor igual a 0. Els temps de vida és igual a ALLOWED_HELLO_LOSS * HELLO_INTERVAL. Per limitar la seva retransmissió per la xarxa, tenen assignat el TTL a 1 salt.

Es defineix un màxim de missatges de *hello* no detectats a la variable ALLOWED_HELLO_LOSS[9]. En aquest cas, si un node ha rebut un *hello* d'un veí durant l'últim DELETE_PERIOD, i no en rep cap altre (o altre tipus de missatge AODV) en el interval de temps ALLOWED_HELLO_LOSS * HELLO_INTERVAL següent, l'enllaç amb aquest veí es considera trencat. Les rutes associades a aquest veí es marquen com invàlides, i el veí deixa de ser considerat com a tal.

Els missatges de *hello* són utilitzats si AODV recau sobre protocols que no poden proporcionar informació sobre l'estat de l'enllaç. El protocol 802.11 pot proporcionar aquesta informació, amb l'algoritme RTS / CTS[10], per exemple. La utilització de reconeixement de dades a nivell d'enllaç pot proporcionar informació sobre el veïnat. Si s'utilitza un protocol capaç de donar aquesta informació, es pot prescindir dels missatges de *hello*.

Una vegada s'ha establert una ruta entre un origen i un destí, el temps de vida associat a aquesta es va actualitzant amb els missatges AODV, mentre la ruta es manté activa.

[8] HELLO_INTERVAL = 1000 ms (per defecte)
[9] ALLOWED_HELLO_LOSS = 2 (per defecte)
[10] Absència de resposta CTS a un RTS

2.2.3.2 Missatges RERR

Si és el node d'origen el que es mou quan s'està transmetent un tràfic, pot repetir el procediment de descobriment de ruta per a la nova situació. En canvi, si és un node intermedi o el destí el que es mou, trencant un o més enllaços, un missatge *Route Error* (RRER) és generat pel node més proper al destí amb connectivitat cap a l'origen. En aquest missatge es guarden la direcció del *next-hop* que ha caigut i les direccions dels destins amb ruta cursada per aquest. Els RERR s'envien només als precursors en les rutes cap a aquest destins. Si només n'existeix un, s'envia el RERR en mode *unicast,* mentre que si n'existeixen més, el RERR s'envia en mode *broadcast,* amb un valor de TTL assignat a 1. Els nodes precursors analitzen el contingut del paquet, i marquen a la seva taula d'encaminament les rutes caps als destins inassolibles com invàlides. No poden ser esborrades totalment fins passat un temps DELETE_PERIOD. Com que a la taula tenen informació sobre els precursors en la ruta cap a aquests destins, es reenvia el missatge RERR fins que arriba a l'origen. Un cop l'origen rep el RERR, pot iniciar un altre cop el procés de descobriment.

Els missatges RERR són enviats als precursors en els següents casos:

- Quan un node detecta que l'enllaç que l'uneix al pròxim salt, en una ruta activa o que està sent reparada, s'ha trencat. S'incrementa en 1 el DSN del destí.
- Quan un node rep dades destinades a un node pel qual no té ruta o està inactiva, excepte en el cas que la ruta estigui sent reparada. En aquest cas s'incrementa en 1 el DSN del destí.
- Quan es rep d'un dels veïns un RERR que afecta a les rutes actives s'ha de reenviar als precursors, si existeixen. Es copia el valor del DSN del missatge a la entrada corresponent de la taula d'encaminament.

El node que detecta el trencament de l'enllaç pot intentar reparar la ruta ell mateix, sense avisar a l'origen amb un RERR. Aquesta opció s'anomena *Local Repair.* Podeu trobar més informació a l'annex A.

2.2.3.3 Si un node es desconnecta i torna a la xarxa

Si un dels nodes que participa en la xarxa ad-hoc es desconnecta i torna a entrar posteriorment, haurà perdut el seu propi número de seqüència, així com els números de seqüència dels altres nodes. Això pot provocar *loops* en el encaminament, ja que altres nodes poden estar utilitzant-lo com a *next-hop* actiu. Per evitar això, el node que acaba d'entrar a la xarxa ha d'esperar un temps DELETE_PERIOD, durant el qual el node actualitza la informació de la seva taula amb els missatges de control que escolta, però no en respon a cap. Si el node rep algun paquet de dades i ell no és el destí, informa a qui li hagi enviat amb un RERR. Finalitzat aquest període, és segur que la resta de nodes de la xarxa hauran esborrat qualsevol informació associada a ell. Un cop finalitzat el període, el funcionament torna a ser el normal.

2.3 Paràmetres i valors per defecte del RFC

En aquest apartat exposem els valors per defecte de les variables del protocol AODV. Les més importants ja han estat esmentades en els apartats anteriors, però a continuació oferim una taula de totes elles:

Taula 2.1 Configuració per defecte de AODV

ACTIVE_ROUTE_TIMEOUT	3000 ms
ALLOWED_HELLO_LOSS	2
BLACKLIST_TIMEOUT	RREQ_RETRIES * NET_TRAVERSAL_TIME
DELETE_PERIOD	K * max (ACTIVE_ROUTE_TIMEOUT, ALLOWED_HELLO_LOSS * HELLO_INTERVAL), amb K=5
HELLO_INTERVAL	1000 ms
LOCAL_ADD_TTL	2
MAX_REPAIR_TTL	0.3 * NET_DIAMETER
MIN_REPAIR_TTL	Últim nombre de salts conegut per al destí
MY_ROUTE_TIMEOUT	2 * ACTIVE_ROUTE_TIMEOUT
NET_DIAMETER	35
NET_TRAVERSAL_TIME	2 * NODE_TRAVERSAL_TIME * NET_DIAMETER
NEXT_HOP_WAIT	NODE_TRAVERSAL_TIME + 10
NODE_TRAVERSAL_TIME	40 ms
PATH_DISCOVERY_TIME	2 * NET_TRAVERSAL_TIME
RERR_RATELIMIT	10
RING_TRAVERSAL_TIME	2 * NODE_TRAVERSAL_TIME * (TTL_VALUE + TIMEOUT_*BUFFER*)
RREQ_RETRIES	2
RREQ_RATELIMIT	10
TIMEOUT_*BUFFER*	2
TTL_START[11]	1
TTL_INCREMENT	2
TTL_THRESHOLD	7

2.4 Implementacions AODV

Són moltes les implementacions que existeixen del protocol AODV. Trobem bàsicament versions pels sistemes operatius Windows, Linux, i altres preparats per dispositius més limitats, com els sistemes operatius del tipus Lineo Embedix OS.

Justificarem l'elecció de la implementació que emprarem en aquest projecte en funció dels punts del RFC que compleixi cada una d'aquests implementacions, quins punts permet modificar, del sistema operatiu utilitzat a les nostres proves i dels resultats obtinguts d'aquestes, com veurem més endavant.

Aquestes són les implementacions d'AODV més rellevants a dia d'avui:

[11] Si s'utilitzen els missatges de *hello*, s'aconsella assignar-li 2

- AODV for Windows v0.1.14 (S.O: Windows Xp)
  - Creada per Intel. Només funciona amb Windows Xp.
  - Només compleix els requeriments MUST del RFC 3561.
  - No suporta *Expanding Ring Search* ni *multicast*, entre d'altres.
  - Homepage: http://moment.cs.ucsb.edu/AODV/aodv-windows.html

- Win AODV University of Bremen v0.1 (S.O: Windows Xp)
  - Compleix el RFC 3561.
  - No funciona del tot correctament i només funciona amb IPv4.
  - Homepage: http://www.aodv.org

- AODV-UU Uppsala University v0.81/0.9 (S.O: Linux)
  - Funciona correctament amb kernels de Linux 2.4.x i 2.6.x. Compatible amb Network Simulator 2 [7]. Escrit en C.
  - Compleix el RFC 3561, amb ports per IPv6 i multicast. Funcions de *gateway*. Suporta informació d'estat de l'enllaç amb driver especial, com HostAP. Pot ser crosscompilat per ser utilitzat en una PDA.
  - Treballa al nivell d'aplicació o espai d'usuari, no com a part integrada del kernel. La versió 0.9 ha aparegut recentment. Treballa com a part del kernel, però pot ser inestable, ja que s'ha rescrit gran part del codi i no està tan testejat com la versió 0.8.1.
  - Homepage: http://core.it.uu.se/AdHoc/AodvUUImpl

- Kernel AODV (S.O: Linux)
  - Implementació del NIST (*National Institute of Standards and Technology* [8] . Compleix el RFC 3561. Escrit en C.
  - Suport a partir del kernel 2.4, amb funcions de *gateway.*
  - Homepage: http://w3.antd.nist.gov/wctg/aodv_kernel/

- JAdhoc University of Bremen v0.2 (S.O: JVM)
  - Funciona amb Windows Xp i Linux, sota Java Virtual Machine.
  - Compleix el RFC 3561.
  - Existeix una versió pel sistema operatiu Embedix de Zaurus, que també pot funcionar en algunes iPaq. Aquesta versió ha estat testejada per nosaltres mateixos, i hem tingut certs problemes que comentarem a l'annex B.
  - Homepage: http://www.aodv.org/

- Tiny AODV (S.O: Tiny OS [9])
  - Els missatges RREP només els genera el destí.
  - No compleix el RFC 3561.
    - Les rutes mai expiren i només s'usa la mètrica de nombre de salts.
    - No existeixen els missatges de *hello* (les rutes no expiren).
  - Homepage: http://cvs.sourceforge.net/viewcvs.py/tinyos/tinyos-1.x/ contrib/hsn/

En aquest estudi, utilitzarem la distribució Fedora Core 2 amb un kernel 2.6.9, que és l'última versió estable a dia d'avui. Treballarem amb Linux, i no Windows, per la seva estabilitat i capacitat de configuració. S'ha escollit la implementació AODV de la Universitat de Uppsala, Suècia (veure [10]).

Aquesta implementació està preparada per treballar amb el kernel 2.6.9, és molt fàcil d'instal·lar i permet configurar tots els paràmetres associats a l'AODV. Existeixen moltes simulacions realitzades amb aquest codi, i les proves són satisfactòries. A més, està escrit en llenguatge C, el que ens donarà més facilitat per entendre'l, tot i que no és l'objectiu d'aquest estudi. A les nostres proves, utilitzarem el mètode de detecció amb missatges de *hello*.

El programador del codi font és en Erik Nordström, qui ha portat la versió a NS-2 és en Björn Wiberg, i el projecte ha estat dirigit per en Henrik Lundgren. Tots ells pertanyen a la Universitat de Uppsala. Peter Lee s'ha encarregat de dur el codi a IPv6, i la Universitat de Maryland s'ha encarregat de la versió *multicast*.

CAPÍTOL 3 ELS PROTOCOLS DE TRANSPORT

En els escenaris ad-hoc, el TCP degrada molt el seu rendiment, com veurem en aquest capítol. Veurem alguna de les solucions plantejades, com el protocol ATCP. Aquesta solució consisteix en afegir una petita capa entre IP i TCP. La capa ATCP s'encarregarà d'escoltar missatges de la xarxa, com els ICMP i els ECN, i actuar en conseqüència, millorant així el funcionament del protocol de transport. La implementació, però, no ha pogut ser instal·lada correctament, com s'indicarà a l'annex C.

3.1 Introducció als problemes de TCP en escenaris ad-hoc

En aquest apartat veurem les causes principals del mal funcionament del protocol TCP en escenaris ad-hoc.

3.1.1 Errors de bit i canvis de ruta

A causa dels error de bit, els paquets es corrompen. El resultat són segments de TCP o paquets de reconeixement (ACKs) perduts. Quan això succeeix repetidament, es crida al control de congestió, donant com a resultat una degradació important de l'ample de banda disponible. Mètodes de correcció de errors no són útils per als escenaris ad-hoc, ja que afegim *overhead* amb un ample de banda limitat, quan potser no és necessari.

Els canvis de ruta afectaran en la mateixa manera. Si hi ha trencaments en els enllaços, pot expirar també el temporitzador de retransmissió (RTO) de TCP durant el període de descobriment de la nova ruta, trobant-nos un altre cop en la situació anterior. Aquest temporitzador es duplica amb cada retransmissió, i pot arribar a un valor màxim de 64 segons. Pot ser que durant aquest període en el que no podem retransmetre s'obtingui la nova ruta, que no podrà ser utilitzada fins l'expiració del RTO.

De la mateixa manera, hem de tenir en compte el concepte de finestra de congestió. Aquesta es defineix en funció de la informació obtinguda de la xarxa mitjançant els RTOs i els ACKs duplicats. Pot ser que una finestra estimada per a una antiga ruta sigui del tot incorrecte per a la nova ruta calculada.

3.1.2 Efectes de l'encaminament multicamí

Si el TCP recau sobre un protocol d'encaminament com TORA (*Temporarilly Ordered Routing Algorithm*), podem tenir problemes semblants. Aquest tipus de protocols guarden més d'una ruta per a una parella orígen-destí donada. Això moltes vegades significa que els paquets arriben desordenats al destí, que respon amb ACKs duplicats, invocant així un altre cop al control de congestió.

3.2 Solució proposada per ATCP

ATCP és una solució que té com a principals característiques:

- No modifica el TCP/IP original.
- Nodes amb ATCP i amb TCP estàndard poden interactuar entre ells. Els nodes sense ATCP no en treuen cap benefici, però.
- ATCP no interacciona en connexions TCP establertes entre un node a la xarxa amb cable i un node a la xarxa sense fils.

A mode resum, aquestes són els trets que diferencien ATCP del TCP:

- Si es detecten errors de bit, deduïts a partir de les expiracions dels RTO i els ACKs duplicats, i s'ha de retransmetre, la retransmissió es fa des de ATCP i no TCP. D'aquesta manera no s'invoca el control de congestió.
- Si hi ha un canvi de ruta o partició a la xarxa, rebrem un missatge ICMP (*Internet Control Message Protocol*) informant-nos que no podem arribar al destí. Aleshores es posa el TCP en *persist mode*. La comunicació TCP es torna a posar en marxa quan rebem resposta del destí, sense invocar el control de congestió.
- Si hi ha un canvi de ruta, el valor de la finestra de congestió s'inicialitza a 1. Pot ser que el valor calculat antigament no sigui adequat per les característiques de la nova ruta.
- Si els paquets arriben desordenats, és ATCP qui els reordena. Així no es generen ACKs duplicats.
- Només en el cas que es rebi un ECN (*Explicit Congestion Notification*) [11], s'invoca el control de congestió.

D'aquesta manera, veiem com ATCP és capaç de aturar, modificar, o actuar en el lloc de TCP en funció de la situació a la que ens trobem. Per distingir les particions de la congestió, depenem dels missatges ICMP *Host Unreachable* i els ECN. Deduirem que hi ha pèrdues al canal davant la expiració d'un RTO o la recepció del tercer ACK duplicat, per a un segment determinat. El funcionament no ha de ser el mateix en els diversos casos. Només quan el funcionament és dolent per culpa de la quantitat de tràfic a la xarxa, es crida el control de congestió a l'origen, actuant el TCP de manera estàndard. A les altres situacions, modifiquem aquest comportament, per tal que millori el rendiment.

3.3 Com funciona ATCP

ATCP funciona d'acord amb un diagrama d'estat amb 4 estat possibles (fig. 3.1). Quan s'inicia una comunicació TCP, ATCP es troba en estat *normal* i és un element invisible. Entrarem en els altres modes de funcionament en funció dels missatges ICMP i ECN rebuts, o davant l'expiració d'un RTO o la recepció del tercer ACK duplicat.

Aquest són els estats possibles de ATCP:

- Estat *normal*: ens trobem en aquest estat quan s'estableix la connexió TCP. En aquest estat, ATCP actua de manera invisible, però es monitoritza la connexió TCP. En el cas de que rebem un ICMP o un ECN canviarem directament d'estat (a *desconnectat* o *congestionat* respectivament). Si no rebem cap d'aquests missatges, només monitoritzarem el nombre d'ACKs duplicats rebuts. En el cas que rebem el tercer ACK duplicat per a un segment determinat, o bé estigui a punt d'expirar el temporitzador de retransmissió, ocultarem aquesta informació al TCP i el posarem en *persist mode*. ATCP canvia a l'estat de *canal amb pèrdues*.

  Quan TCP es troba en *persist mode*, envia *probe packets* cap al destí cada certs intervals de temps. Quan el destí rep un d'aquest paquets (es restableix la connexió), contesta amb més dades o amb un ACK duplicat.

- *Canal amb pèrdues*: ATCP canvia a aquest estat quan està a punt d'expirar el RTO o es rep el tercer ACK duplicat per a un segment determinat. En aquest estat, ATCP retransmet els segments no reconeguts del *buffer* de TCP, amb uns temporitzadors propis, mentre aquest es troba en estat *persist mode*. Una vegada es rep un ACK per a un d'aquests segments, l'ATCP el reenvia a la capa TCP. Aleshores TCP surt del *persist mode* i l'ATCP torna a l'estat *normal*.

- *Congestionat*: per poder gaudir d'aquesta característica, cal que els nodes de la xarxa tinguin suport per activar el *flag* ECN dels segments TCP, en el cas que detectin congestió. Si es rep un segment amb aquest flag activat, l'ATCP entra en estat *congestionat* immediatament, i TCP invoca el control de congestió. ATCP deixa de monitoritzar la connexió TCP i actua de manera totalment invisible. Quan TCP transmet un nou segment, l'ATCP tornarà al estat *normal*.

- *Desconnectat*: els nodes de la xarxa ad-hoc poden trigar un cert temps en trobar una nova ruta per a un destí determinat. Durant aquest temps, es perden segments i els respectius ACKs, que són transmesos inútilment. A més, s'invoca el control de congestió. Per solucionar això, l'ATCP actua quan rep un missatge ICMP *Host Unreachable*. Si es rep un d'aquests missatges, ATCP entra en mode desconnexió, i TCP en *persist mode*, congelant el seu estat. Una vegada el destí contesta amb més dades o amb un ACK duplicat, l'ATCP torna a l'estat *normal* i el TCP torna a l'estat anterior al tall. En el cas que la ruta sigui diferent a la utilitzada anteriorment, s'assigna un valor de 1 segment a la finestra de congestió. Així, aquesta creixerà adequadament per a les característiques del canal ràdio de la nova ruta.

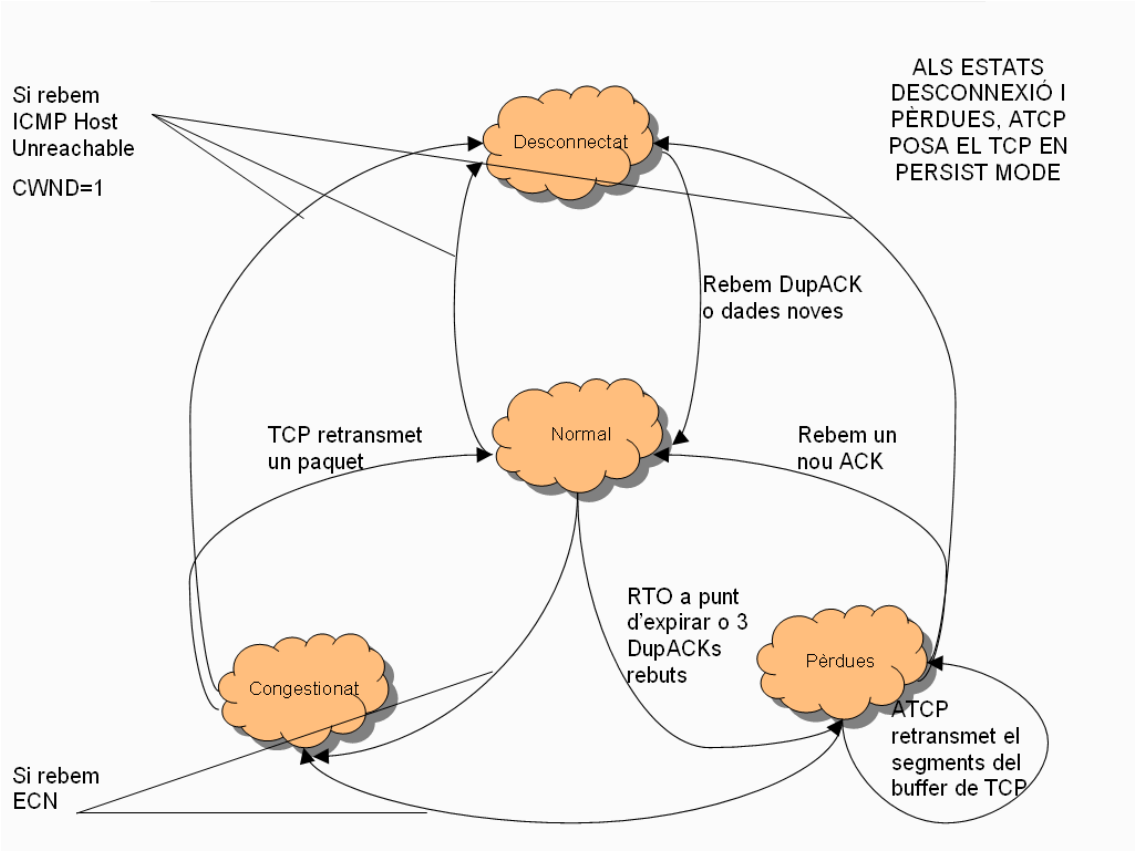


Fig. 3.1 Diagrama d'estats del protocol ATCP.

3.4 Conclusions

L'ATCP és una solució original per afrontar el problema del TCP en les xarxes ad-hoc. Podem observar que l'avantatge principal radica en que el protocol és capaç d'identificar la causa dels problemes que apareixen al llarg de la transmissió i actuar en conseqüència. ATCP manté la finestra de congestió en el cas de tenir pèrdues degut a errors de bit en el canal, evitant que es cridi el control de congestió de TCP. Per tant, en aquells casos on el problema principal sigui un elevadíssim error de bit i una elevada freqüència en les particions de la xarxa ad-hoc, l'ATCP ens donarà un major rendiment que TCP, mentre que en escenaris on el problema sigui la congestió, això no succeirà. En el cas de particions de xarxa i canvis de ruta, a part de no invocar el control de congestió no saturarem més el canal de transmissió amb paquets que no podran arribar al destí, ja que en el moment que un RTO expiri, es congelarà l'estat de TCP, evitant que el RTO creixi inútilment, i només s'enviaran paquets *probe* a la xarxa.

Malgrat que s'ha intentat estudiar el rendiment d'una implementació real de l'ATCP, a la pràctica no hem pogut veure aquests resultats, ja que la única implementació existent de l'ATCP no ha pogut ser instal·lada ni amb l'ajuda de un dels coautors del protocol. Podeu trobar més detalls d'aquest problema a l'annex C.

CAPÍTOL 4 INSTAL·LACIÓ I CONFIGURACIÓ DELS DISPOSITIUS DE LA XARXA AODV

En aquest capítol veurem la instal·lació i configuració de la xarxa ad-hoc AODV real avaluada en aquest projecte. Farem servir el sistema operatiu Linux, concretament la distribució Fedora Core 2 i el kernel 2.6.9. La xarxa ad-hoc AODV estarà formada per un grup de portàtils amb el sistema operatiu Linux. Amb l'objectiu de fer les proves amb el protocol ATCP, utilitzarem un PC amb el sistema operatiu FreeBSD 4.2, ja que només existeix una implementació per a aquesta versió. Les comunicacions seran encaminades al llarg de la xarxa ad-hoc pel protocol AODV, a través d'un dels portàtils que farà les funcions de *gateway* entre les dues xarxes.

4.1 Dispositius i software utilitzats

A continuació es detalla el maquinari utilitzat en els nostres escenaris i les seves característiques:

- 4 portàtils Acer 662LCi, equipats amb una targeta sense fils Intel Pro/Wireless 2100 (miniPCI interna), que compleix l'estàndard 802.11b de la IEEE (11Mbps). Utilitzats en totes les proves.

- 1 portàtil Toshiba Satellite, equipat amb una targeta sense fils PCMCIA Cisco, que també compleix l'estàndard 802.11b. Utilitzat com a cinquè portàtil en les proves de demora de descobriment de ruta i ample de banda consumit per AODV.

A tots els portàtils hi ha instal·lada la distribució Linux Fedora Core 2 [12]. Per defecte, aquesta distribució ve dotada amb el kernel 2.6.6-1.435.2.3, però s'ha actualitzat al kernel 2.6.9, la última versió estable en el moment de fer aquestes proves. Aquesta distribució porta instal·lat per defecte el paquet *Wireless Tools* [13], que conté les eines, com *iwconfig*, que ens serviran per configurar la xarxa sense fils, i drivers per a una multitud de targetes 802.11.

Per a la targeta Cisco, hem utilitzat els drivers inclosos al kernel, però aquest no inclou suport per les targetes Intel Pro/Wireless. Hem utilitzat la versió 1.01 del driver IPW2100. Tots els portàtils porten instal·lada la versió 0.81 del AODV-UU, del que ja hem parlat en el segon capítol.

L'escenari en el que s'haurien fet les proves amb ATCP el farem servir igualment per veure les diferències entre les implementacions de TCP de Linux i FreeBSD [14]. Utilitzarem la versió 4.2 de FreeBSD, ja que la implementació de ATCP només funciona, en teoria, amb aquesta versió en concret del sistema operatiu.

4.2 Configuració dels dispositius de la xarxa ad-hoc

4.2.1 Instal·lació dels drivers

El kernel 2.6.9 no inclou suport per a les targetes Intel PRO/Wireless instal·lades als portàtils. Existeixen uns drivers de codi obert per a Linux, anomenats IPW2100 [15]. És tracta d'un projecte iniciat per Intel, però en desenvolupament constant per la comunitat internauta. En un principi, per a les proves es va utilitzar la versió del kernel 2.6.6-1.435.2.3, inclosa per defecte en Fedora Core 2, amb la versió 0.62 dels drivers IPW2100. Es va aconseguir instal·lar la xarxa ad-hoc, i vam fer proves per comprovar la connectivitat a nivell IP amb *ping*. Vam tenir problemes amb la recepció del senyal i en el funcionament global, obtenint una quantitat molt alta de pèrdua de paquets i observant grans canvis en el comportament del sistema en funció del moviment dels portàtils. Al instal·lar la versió 2.6.9 del kernel amb la versió 1.01 del driver per les targetes, aquests problemes van desaparèixer.

Els drivers es poden compilar com a part integrada del kernel o com un mòdul. En aquest estudi, hem decidit instal·lar el driver com un mòdul que carregarà el kernel quan li ho demanem. El primer que hem de fer és descarregar el codi font del driver. Aquest es troba comprimit en format .tgz. Per descomprimir el paquet, podem utilitzar les eines del entorn gràfic de Linux (KDE o GNOME, veure [16]), o bé obrir un terminal, anar al directori on es troba l'arxiu i executar:

```
# tar –zxvf ipw2100-1.0.1.tgz
```

Una vegada hem desempaquetat i descomprimit l'arxiu, podem veure les instruccions d'instal·lació en l'arxiu README. Els passos per instal·lar el driver són els següents. Es necessita tenir instal·lat i configurat el compilador *gcc* [17]. Obrim un terminal i ens situem dins el directori dels drivers, executant:

```
# make
# make install
```

Amb la primera comanda compilarem el codi font per obtenir el mòdul, i amb la segona guardarem aquest últim en el directori corresponent (/lib/modules/linux-2.6.9). El driver no funcionarà si no instal·lem el firmware adequat. Aquest el podem obtenir de la mateixa pàgina del projecte IPW2100. El firmware consta de tres arxius comprimits. Per descomprimir-los, copiarem l'arxiu al directori /usr/local/hotplug/firmware (si no existeix el directori, l'haurem de crear), i executar en un terminal:

```
# tar –zxvf ipw2100-fw-1.3.tgz
```

Una vegada hem realitzat aquest passos, podem procedir a carregar el mòdul, i per tant el driver, executant:

```
# modprobe ipw2100
```

Per a la targeta Cisco, podem carregar el mòdul directament[12] executant:

#modprobe airo_cs

Si hi hagués algun problema amb aquest mòdul, caldria comprovar que està activat a la configuració del kernel. Per més informació sobre com funciona i com es pot configurar el kernel de Linux, consultar l'annex D.

4.2.2 Configuració de les targetes 802.11b

Hem aconseguit que el sistema operatiu reconegui la targeta sense fils, però ara hem de configurar la xarxa en mode ad-hoc. Aquests passos ens serviran tant per configurar les targetes Intel PRO/Wireless com la PCMCIA. Podem fer-ho en mode gràfic o bé amb un terminal. Gràficament és més fàcil, però l'eina de configuració no funciona sempre del tot correctament, sobretot en la detecció dels dispositius. Primer de tot, recomanem esborrar qualsevol configuració de xarxa, és a dir, qualsevol dispositiu i les xarxes associades. Seguint l'ordre dels passos aquí detallats, obtindrem *eth0* com a identificador del dispositiu *wireless*. Més endavant, haurem d'activar la targeta Ethernet en un dels portàtils. Se li assignarà l'alies *eth1*. Assumirem en endavant que així s'ha realitzat la configuració de la xarxa.

Per configurar la xarxa ad-hoc amb la interfície gràfica, utilitzarem la utilitat *Control de dispositivos de red*, al menú *Herramientas del sistema*. Des d'un terminal, obrirem la mateixa finestra executant la comanda *neat.* Des d'aquesta aplicació, podem activar (Nuevo), desactivar (Borrar), i configurar (Modificar) els dispositius Ethernet i wireless dels nostres portàtils.

Fig. 4.1 Pantalla de configuració dels dispositius.

[12] Sempre que s'hagi habilitat el mòdul a l'hora de compilar el kernel.

Els menús són molt intuïtius. Recordem que es recomana esborrar qualsevol configuració existent, seleccionant tots els dispositius que apareixen a les pantalles de *Dispositivos* i *Hardware*, clicant a la icona *Borrar* i aplicant els canvis. Així, assegurem que els serveis de xarxa són reiniciats, i que les configuracions i dispositius anteriors són esborrats. Per donar d'alta un dispositiu, carregarem els mòduls corresponents amb *modprobe*. Aleshores anirem al menú *Nuevo* a *Hardware*.

Una vegada haguem configurat de nou els dispositius tal i com desitjàvem (*eth0* Intel PRO/Wireless o Cisco per la interfície 802.11b i *eth1* Broadcom per la Ethernet), procedim a clicar a *Nuevo* a la part corresponent a *Dispositivos*, per tal de configurar la xarxa.

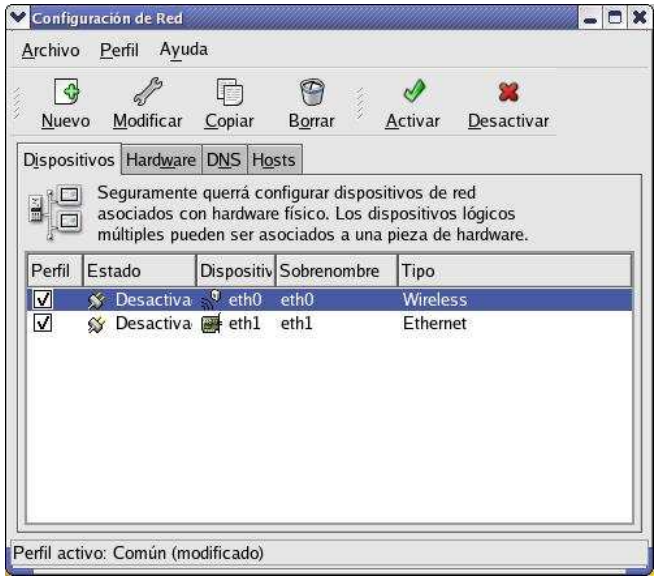


Fig. 4.2 Pantalla principal de la eina gràfica de configuració de xarxa.

Al primer pas se'ns demana quin és el tipus de dispositiu que volem configurar. Escollirem l'apartat de dispositius sense fils. En el segon pas, si hem instal·lat i carregat correctament els drivers, apareixeran els dispositius *Pro/Wireless LAN 2100 3B Mini PCI adapter* pels portàtils Centrino i *airo_cs* per la tarjeta Cisco.

Al tercer pas, configurem la xarxa sense fils. Escollirem el mode ad-hoc. Hem de configurar un identificador de xarxa, escollir un canal, una velocitat de transmissió i si volem, una clau de xifrat, que serà utilitzada per codificar les dades. Tots quatre valors han de ser idèntics a tots els portàtils perquè tot funcioni correctament. La velocitat la establirem al màxim possible amb l'estàndard 802.11b, 11 Mbps. Utilitzarem xifrat WEP de dades (veure [18]), ja que així evitarem que altres puguin espiar les nostres comunicacions. És un mètode del que ja es coneixen molts punts febles, però les nostres proves no necessiten més seguretat que aquesta. Per al canal, podem escollir un valor de 1 a 14. Un canal és una fracció del medi ràdio que usarem per no interferir altres comunicacions d'altres xarxes. Per saber quines són les xarxes que tenim al voltant, podem executar des d'un terminal:

iwlist eth*X* scanning

On *X* es el valor de la interfície sense fils (serà eth0 si no hem configurat cap interfície abans, com per exemple, la Ethernet). Les targetes utilitzades en aquest estudi suporten un escanejat de les freqüències del canal ràdio. Amb aquesta comanda, podem veure quines xarxes podrien interferir amb la nostra. Es mostrarà un llistat de totes elles i en quin canal estan treballant. Per evitar les interferències, és recomanable situar-se almenys a quatre canals de distància de qualsevol d'elles, encara que si la intensitat del tràfic a les xarxes és baixa, podem disminuir la distància. Per a finalitzar, al quart pas de l'assistent, seleccionarem l'adreça IP de la interfície i la màscara de subxarxa. És important que tots els equips es trobin a la mateixa subxarxa per a que AODV-UU funcioni correctament. En el nostre banc de proves, hem seleccionat la subxarxa 192.168.7.x, amb màscara 255.255.255.0. No definirem cap *gateway*, perquè no estudiarem la possible sortida a altres xarxes, com Internet, utilitzant AODV (funció implementada en estat experimental a AODV-UU). Una vegada hem finalitzat els quatre passos, accedim a una última pantalla amb el resum de la configuració. Si acceptem, els nous dispositius seran activats i la xarxa ad-hoc estarà correctament configurada.

A continuació, detallarem quines són les comandes per efectuar en mode text les operacions anteriors, juntament amb una breu explicació. Es recomana el ús d'aquesta opció, sobretot amb l'ajuda de *scripts*. Per a més informació, recomanem consultar els manuals que acompanyen els programes de les distribucions Linux. Bàsicament, utilitzarem les comandes *ifconfig*, amb la que configurarem els paràmetres IP, com l'adreça IP i la màscara de xarxa, i *iwconfig*, amb la que establirem les característiques de la nostra xarxa sense fils. Utilitzarem aquest *scripts* per configurar la xarxa ràpidament i fer les diferents proves. Podem disposar d'un conjunt de *scripts* que ens permetin canviar d'una configuració a una altra ràpidament. Consultar l'annex E per veure els *scripts* que s'han utilitzat en les diferents proves d'aquest estudi. Afegirem també les línies necessàries perquè es carreguin els mòduls corresponents als nostres dispositius. Aquestes són les comandes a executar:

modprobe ipw2100 (només als portàtils Centrino)
modprobe airo_cs (només al portàtil amb PCMCIA Cisco)
ifconfig eth0 192.168.7.x netmask 255.255.255.0 (configuració IP)
iwconfig eth0 mode Ad-Hoc (per treballar en mode ad-hoc)
iwconfig essid *X* (*X* serà el identificador de la nostra xarxa ad-hoc)
iwconfig eth0 channel *X* (escollim la nostra banda de freqüència)
iwconfig eth0 key s:*X* (habilitem l'encriptació WEP, X és la clau de xifrat)

Una vegada executades les comandes anteriors, podem veure que els mòduls s'han carregat correctament amb *lsmod*. Per revisar la configuració IP utilitzarem *ifconfig*, i per revisar la configuració de la xarxa ad-hoc utilitzarem *iwconfig*. Si alguna de les opcions no apareix amb el valor correctament configurat, tornar a esborrar tots els dispositius de xarxa, descarregar[13] els mòduls del kernel i tornar a repetir els passos anteriors. És molt fàcil que això succeeixi utilitzant l'assistent gràfic, per això recomanem les comandes en mode text conjuntament amb l'ús de *scripts*.

[13] Utilitzar modprobe –r *mòdul*

4.2.3 Configuració de les targetes Ethernet

Els portàtils Acer venen dotats d'una targeta Ethernet Broadcom. Els drivers d'aquesta targeta també estan inclosos a la distribució Fedora Core 2. Recordem que un dels portàtils actuarà com a *gateway* entre la xarxa AODV i el PC amb ATCP. Si hem seguit els passos esmentats en aquest capítol, aquest dispositiu haurà d'obtenir el àlies *eth1.* Per activar el mòdul corresponent a aquest dispositiu, executarem en el terminal del portàtil que actua com a *gateway* (o afegirem als scripts anteriors):

```
# modprobe b44
# ifconfig eth1 192.1687.6.x netmask 255.255.255.0
```

Hem escollit 192.168.6.x com a identificador d'aquesta subxarxa.

FreeBSD també inclou els drivers necessaris per fer funcionar la targeta Ethernet. Recordem que FreeBSD i Linux provenen de UNIX. *Ifconfig* és una de les eines que els dos sistemes operatius han heretat del seu antecessor. Podem aleshores deduir que a FreeBSD es configurarà el dispositiu executant:

```
# ifconfig eth0 192.1687.6.x netmask 255.255.255.0
```

No cal carregar cap mòdul, ja que la instal·lació detecta el dispositiu i es carrega automàticament. Se li assignarà el àlies *eth0* per defecte.

4.3 Instal·lació i funcionament de AODV-UU

Primer de tot hem de descarregar el codi font del programa des de la web del projecte a la Universitat de Uppsala. Trobarem que les dues últimes versions disponibles són la 0.81 i la 0.9. A la mateixa pàgina se'ns recomana que s'utilitzi la versió 0.81. Aquesta versió treballa com un dimoni que escolta els missatges AODV, mentre que a la nova versió 0.9 s'està intentant integrar el protocol dins el mateix kernel. A la nova versió s'ha rescrit gran part del codi font, i per això no és del tot recomanable, ja que la versió anterior està molt més testejada. Una vegada tenim el codi font, hem de descomprimir-lo. Obrim un terminal, ens situem al directori on haguem instal·lat la implementació de Uppsala, i executem la comanda per desempaquetar i descomprimir:

```
# tar –zxvf aodv-uu-0.8.1.tar.gz
```

En el interior d'aquest fitxer, hi ha un directori que conté els arxius README i INSTALL entre d'altres, que gairebé sempre acompanyen el software per Linux, i es recomana llegir abans d'instal·lar qualsevol programa. En el arxiu params.h es troben definides totes les variables que podrem configurar amb aquesta implementació, per realitzar les nostres proves. Podem canviar el valor de les variables amb un editor com *vi,* en mode text, o utilitzar un editor gràfic com *gedit*. Una vegada haguem configurat AODV-UU, podem dur a terme la instal·lació, executant:

```
# make && make install && make clean
```

A Linux es poden concatenar les ordres amb l'operador &&. D'aquesta manera, compilarem el codi amb les opcions definides a params.h i instal·larem el mòdul corresponen a l'AODV, netejant finalment els arxius temporals de la operació.

Si hem executat totes les operacions anteriors correctament, per executar el dimoni aodvd, obrim un terminal i executem, des de qualsevol directori:

```
# aodvd
```

A la figura 4.3 es mostra com arrenca el dimoni AODV, deshabilitant el funcionament dels *hello* per defecte (opció -o).

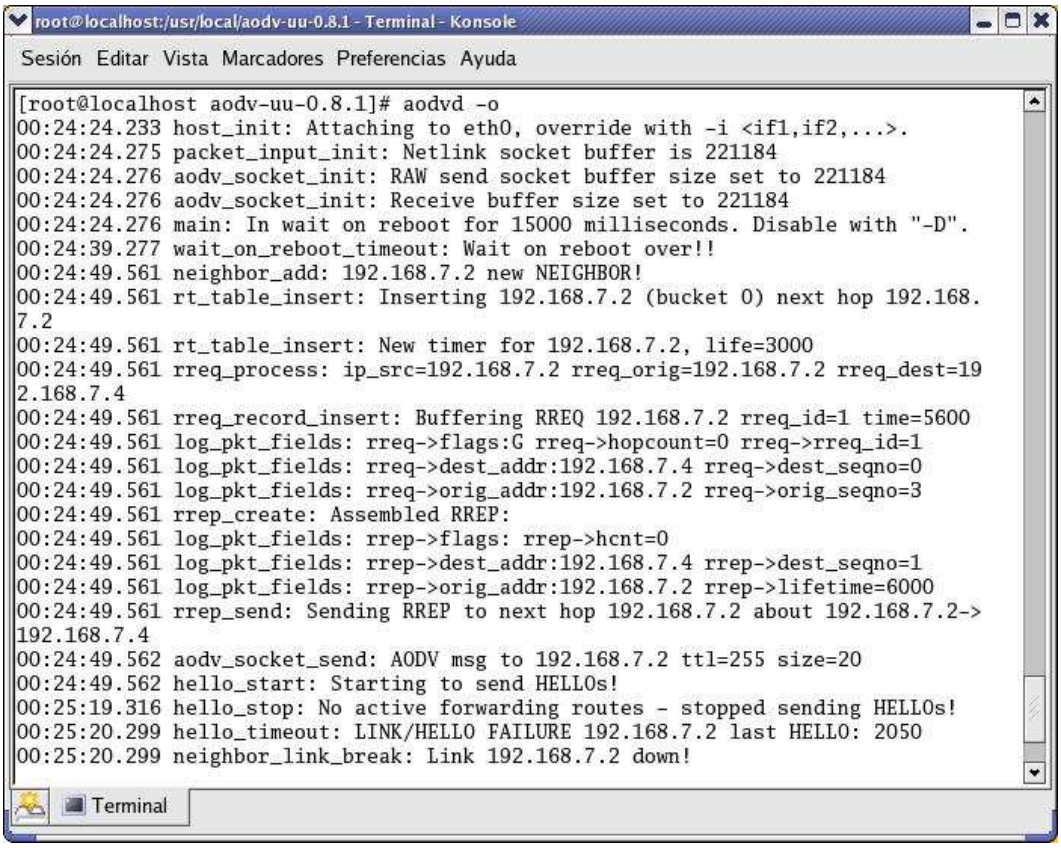


Fig. 4.3 Captura d'un terminal executant el dimoni AODV de Uppsala

Com podem veure, els primers 15 segons pertanyen al període de wait-on-reboot, comentat més endavant. Passats aquest 15 segons, si el node rep un missatge de *hello* o un RREQ d'un dels nodes adjacents, el considera veí (neighbor_add), i l'insereix a la taula d'encaminament (rreq_record_insert). Si coneix un destí vàlid per una ruta demanada, generarà un missatge RREP (rrep_create) amb direcció a l'origen. Des del moment que s'estableix la transferència d'informació, el node comença a enviar missatges de *hello* a la freqüència definida a la variable HELLO_INTERVAL (per defecte 1 segon), mentre la ruta segueix activa. Quan el node d'origen atura la transmissió, deixa

d'enviar missatges de *hello*, així com tots els nodes de la ruta. Quan detectem la pèrdua d'un nombre de missatges de *hello* d'un dels nostres veïns igual a la definida a la variable ALLOWED_HELLO_LOSS, considerem l'enllaç trencat (*LINK FAILURE*), i el veí deixa de ser-ho. Les rutes inutilitzades s'esborren passat un DELETE_PERIOD.

Al programa se li poden passar paràmetres per modificar el seu comportament. A continuació en destaquem els més importants :

- -h: llista totes les opcions.
- -o: els nodes només enviaran *hello*s quan tinguin una ruta activa. Si no activem aquesta opció, els nodes començaran a enviar missatges de *hello* tant aviat arranquem el dimoni, encara que no hi hagi tràfic *on-demand* a la xarxa. No és del tot lògic que el programa operi per defecte trencant una norma bàsica del RFC.
- -D: deshabilita el wait-on-reboot. Si no activem aquesta opció, el node processarà però no contestarà els missatges AODV, fins que no expirin DELETE_PERIOD segons des de l'arrencada del dimoni.
- -d: executa el programa com un dimoni independent[14] del terminal.
- -x: deshabilita *Expanding Ring Search*. D'aquesta manera, el primer dels missatges RREQ arriba a tots els nodes de la xarxa.
- -w: suport de *gateway* (experimental).
- -l: escriu un log al fitxer /var/log/aodvd.rtlog
- -f: habilita la opció de rebre informació de l'estat de l'enllaç[15] directament d'aquesta capa.

Aquest és el valor per defecte de les variables principals definides al fitxer params.h (només les variables que es refereixen a la utilització de missatges de *hello*, no *link layer feedback*):

ACTIVE_ROUTE_TIMEOUT	3000 (ms)
TTL_START_HELLO	2
TTL_INCREMENT	2
TTL_THRESHOLD	7
ALLOWED_HELLO_LOSS	2
HELLO_INTERVAL	1000 (ms)
MY_ROUTE_TIMEOUT	2 * ACTIVE_ROUTE_TIMEOUT
NEXT_HOP_WAIT	NODE_TRAVERSAL_TIME + 10
RERR_RATELIMIT	10
RREQ_RETRIES	2
RREQ_RATELIMIT	10
TIMEOUT_BUFFER	2
DELETE_PERIOD	K * max(ACTIVE_ROUTE_TIMEOUT, ALLOWED_HELLO_LOSS * HELLO_INTERVAL), amb K=5

Com veiem, es compleixen els paràmetres especificats al RFC. Al utilitzar missatges de *hello* per gestionar la connectivitat local, el valor de TTL_START està inicialitzat a 2 enlloc de 1, tal i com s'aconsella al RFC.

[14] Al tancar el terminal, tancarem també el dimoni AODV, ja que el primer és el procés pare.
[15] Es necessita un driver addicional especial, com HostAP.

CAPÍTOL 5 ESQUEMES DE PROVES PER A L'ANÀLISI DEL RENDIMENT DE XARXES AD-HOC AMB AODV

En aquest capítol exposarem i descriurem els escenaris emprats a les proves d'aquest estudi. La definició exacta de les proves i l'anàlisi dels resultats obtinguts es durà a terme en el capítol 6.

5.1 Escenaris estàtics

L'escenari bàsic que podem realitzar el formen dos dispositius ad-hoc, comunicats directament entre sí via ràdio. Aquest escenari pot ser realitzat amb l'ajuda de les eines *ifconfig* i *iwconfig*, proporcionades a la distribució Fedora Core 2. Aquest escenari ens serà útil per conèixer les eines principals que farem servir en aquestes proves, i per tenir una primera referència del temps en els que ens mourem.

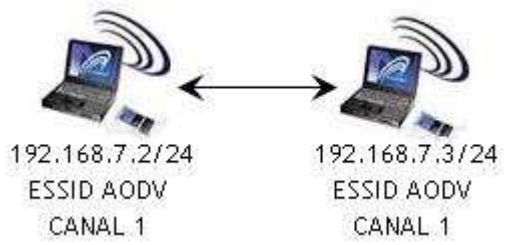


Fig. 5.1 Escenari bàsic de proves

Per realitzar aquest escenari, només cal que els dos portàtils estiguin en un mateix rang de cobertura i que estiguin configurats per treballar en una mateixa xarxa ad-hoc. Els paràmetres que caracteritzen a aquesta són l'ESSID (identificador de xarxa), el canal i la clau de xifrat utilitzats. Utilitzarem els mateixos valors per a aquests paràmetres i utilitzarem la mateixa xarxa IP per als dos ordinadors. Per veure com es realitza aquesta configuració, revisar l'apartat 4.2.2. No haurem de configurar cap ruta ja que existeix visibilitat directa.

Podem afegir més dispositius a aquesta xarxa. Si tots els nodes (n) de la xarxa es troben dins un mateix rang de cobertura, l'escenari té n nodes, però només 1 salt, i així no podem analitzar el rendiment del protocol d'encaminament. El primer que hem de tenir en compte és que al utilitzar el protocol 802.11b, com més nodes comparteixen i competeixen pel medi, més es redueix l'ample de banda disponible. Tots els nodes tenen visibilitat directa entre ells, i no han d'utilitzar encaminament real per transmetre la informació.

Aquest és un problema al muntar els escenaris que descriurem a continuació, utilitzant els equips d'aquestes proves. La potència de transmissió de les targetes Intel PRO/Wireless és força elevada, i els drivers utilitzats, que són els únics disponibles per a Linux, no suporten controlar[16] aquesta potència. S'han fet proves per veure quin és el radi de transmissió d'aquestes targetes, i és impossible muntar els escenaris desitjats, tenint en compte que el que volem és que els nodes només tinguin visibilitat directa amb els nodes adjacents. La potència de transmissió és tan elevada que nodes separats uns 100 metres en línia recta segueixen veient-se l'un a l'altre. A més, en aquest casos, petits moviments del mateixos nodes o de cossos que es mouen en el voltant, provoquen rebots i interferències, donant com a resultat uns escenaris del tot inestables i gairebé impossibles de muntar. S'han utilitzat targetes PCMCIA, de diverses marques, per veure si alguna oferia la possibilitat de controlar la potència, i els resultats han estat igualment negatius.

A les proves s'ha utilitzat l'eina *iptables*[17], inclosa amb la distribució Linux utilitzada. *Iptables* ens servirà per emular els escenaris desitjats, tot i que els equips es trobin en una mateixa zona de cobertura entre tots ells. Tot i això, estaran compartint el mateix medi, amb el que veurem que el ample de banda disponible disminuirà en funció del nombre d'equips que utilitzem. No és un escenari del tot real, però és l'únic que s'ha pogut muntar amb garanties.

Per observar aquest fenomen, configurarem un escenari en línia. Els nodes es trobaran virtualment només a l'abast dels seus nodes adjacents, amb rutes estàtiques configurades prèviament a tots ells. La comunicació serà extrem a extrem, travessant tots els nodes de la xarxa. Amb aquest escenari, podrem caracteritzar el ample de banda disponible en funció del nombre de nodes que formen aquesta. Si definim el nombre de nodes n, obtindrem N salts, que equivalen als n nodes menys 1. Un altre cop, per que tot funcioni correctament, els dispositius han de tenir definit el mateix valor pels diferents paràmetres de la xarxa ad-hoc. Sobre aquest tipus d'escenari farem mesures de retard dels paquets i d'ample de banda disponible extrem a extrem.

Fig. 5.2 Escenari amb 5 nodes i 4 salts

A la taula 5.1 es mostra com s'ha assignat la relació de parelles MAC-IP. Aquests valors seran vàlids per a totes les proves realitzades en aquest estudi.

[16] Es pot fer amb la comanda *iwconfig ethX txpower {-10,+19}*. La comanda no és rebutjada, però no obtindrem els resultats desitjats.
[17] La versió inclosa a Fedora Core 2 és la 1.2.9. Més informació a l'annex F.

A continuació, mostrem quines són les comandes a executar en un terminal per aconseguir que els nodes rebutgin els paquets provinents dels nodes no adjacents.

Taula 5.1 Relació MAC-IP

MAC	IP
192.168.7.2	00:0C:F1:2A:3B:29
192.168.7.3	00:0C:F1:0D:EB:6A
192.168.7.4	00:0C:F1:10:6B:A4
192.168.7.5	00:0C:F1:2A:38:EE
192.168.7.6	00:07:EB:31:80:1B

PC 192.168.7.2
iptables -t filter -A INPUT -m mac --mac-source 00:0C:F1:10:6B:A4 -j DROP
iptables -t filter -A INPUT -m mac --mac-source 00:0C:F1:2A:38:EE -j DROP
iptables -t filter -A INPUT -m mac --mac-source 00:07:EB:31:80:1B -j DROP
PC 192.168.7.3
iptables -t filter -A INPUT -m mac --mac-source 00:0C:F1:2A:38:EE -j DROP
iptables -t filter -A INPUT -m mac --mac-source 00:07:EB:31:80:1B -j DROP
PC 192.168.7.4
iptables -t filter -A INPUT -m mac --mac-source 00:0C:F1:2A:3B:29 -j DROP
iptables -t filter -A INPUT -m mac --mac-source 00:07:EB:31:80:1B -j DROP
PC 192.168.7.5
iptables -t filter -A INPUT -m mac --mac-source 00:0C:F1:2A:3B:29 -j DROP
iptables -t filter -A INPUT -m mac --mac-source 00:0C:F1:0D:EB:6A -j DROP
PC 192.168.7.6
iptables -t filter -A INPUT -m mac --mac-source 00:0C:F1:2A:3B:29 -j DROP
iptables -t filter -A INPUT -m mac --mac-source 00:0C:F1:0D:EB:6A -j DROP
iptables -t filter -A INPUT -m mac --mac-source 00:0C:F1:10:6B:A4 -j DROP

Ara, però, els nodes hauran de reenviar els paquets que arriben dels seus veïns, i aquesta característica no està habilitada per defecte, almenys amb la distribució Linux utilitzada. A més, haurem d'evitar que els nodes enviïn i rebin missatges *ICMP redirect*[18]. Per aconseguir que els nodes siguin capaços de reenviar paquets, i desactivar els missatges *ICMP redirect*, haurem de canviar paràmetres del registre de Linux. Podem modificar el registre amb els fitxers que trobarem al directori /proc/sys, i per configurar els paràmetres de xarxa, ens interessen concretament els directoris /proc/sys/net/ipv4 i /proc/sys/net/ipv4/conf/eth*X* (on *X* és 0 per la interfície *wireless*, i 1 per la interfície Ethernet). Aquests directoris contenen fitxers, el nom dels quals reflecteixen la opció que activen o desactiven. Si volem activar la opció, escriurem el valor 1 en el fitxer, i un 0 si el que volem és desactivar-la, En el nostre cas volem activar el *forwarding* als nodes, és a dir, que puguin reenviar paquets, i desactivar la opció de generar o rebre missatges *ICMP redirect*. Per fer això, tenim dues opcions. Si volem canviar algun d'aquests valors en

[18] Tipus de missatge ICMP enviats per un node intermedi a l'origen, per a que aquest modifiqui la seva ruta cap al destí a una més curta

qualsevol moment, podem obrir un terminal i reescriure els valors dels fitxers de configuració dels directoris que formen el registre de Linux. Aquest són les comandes a executar:

```
# echo 1 > /proc/sys/net/ipv4/ip_forward
# echo 0 > /proc/sys/net/ipv4/conf/all/ accept_redirects
# echo 0 > /proc/sys/net/ipv4/conf/all/send_redirects

# echo 1 > /proc/sys/net/ipv4/conf/ethX /forwarding
# echo 0 > /proc/sys/net/ipv4/conf/ethX/accept_redirects
# echo 0 > /proc/sys/net/ipv4/conf/ethX/send_redirects
```

El primer bloc defineix els valors per defecte de totes les interfícies. En el segon bloc, podem especificar un valor en concret per a una interfície determinada. Si volem que aquestes opcions es carreguin a l'inici del sistema operatiu, podem fer-ho afegint les següents línies al fitxer /etc/sysctl.conf:

```
net.ipv4.ip_forward = 1
net.ipv4.conf.all.send_redirects = 0
net.ipv4.conf.all.accept_redirects = 0

net.ipv4.conf.ethX.forwarding = 1
net.ipv4.conf.ethX.send_redirects = 0
net.ipv4.conf.ethX.accept_redirects = 0
```

Hem de tenir en compte que ara els nodes no tenen visibilitat directa, així que haurem de definir una ruta estàtica per aconseguir la comunicació extrem a extrem. Per establir les rutes estàtiques, farem servir la comanda *route*. Amb aquesta eina podem definir per a un destí determinat, per quina interfície hauran de sortir els paquets dirigits a ell i qui serà el *next-hop*. Trobareu més informació sobre *route* a l'annex G. Aquestes són les comandes a executar:

PC 192.168.7.2: # route add –net 192.168.7.6/24 gw 192.168.7.3

PC 192.168.7.3: # route add –net 192.168.7.6/24 gw 192.168.7.4

PC 192.168.7.4: # route add –net 192.168.7.6/24 gw 192.168.7.5
route add –net 192.168.7.2/24 gw 192.168.7.3

PC 192.168.7.5: # route add –net 192.168.7.2/24 gw 192.168.7.4

PC 192.168.7.6: # route add –net 192.168.7.2/24 gw 192.168.7.5

Amb aquesta configuració, els nodes només podran arribar als nodes adjacents i als inclosos en les comandes anteriors (nodes extrems). Tots els nodes intermedis saben on tenen que enviar la informació destinada als extrems, però els nodes extrems no saben arribar a tots els nodes intermedis (només a aquells que són el *next-hop* cap al destí oposat). Podrem comprovar que els paquets viatgen per l'itinerari desitjat amb la comanda *traceroute*[19].

[19] Aquesta comanda mostra per pantalla els nodes que travessa la informació fins arribar a un destí determinat.

5.2 Escenaris dinàmics

Farem servir els escenaris amb rutes estàtiques per poder caracteritzar el camí extrem a extrem utilitzat. Aquest encaminament, però, no és vàlid en el món ad-hoc. Els dispositius es mouen i això fa variar la topologia de la xarxa. Per poder comunicar-se, necessiten d'un protocol com AODV, capaç de reaccionar a aquests canvis, mantenint un encaminament dinàmic. Aquest tipus de protocols, però, afecten a altres paràmetres de la xarxa. Usant encaminament estàtic, els nodes saben en tot moment on han d'enviar la informació, mentre que utilitzant un protocol d'encaminament reactiu com AODV, si el node origen no coneix la ruta cap al destí, ha de trobar abans una ruta vàlida cap a aquest. El descobriment afegeix una certa latència en el moment d'enviar la informació. A més, utilitzant protocols d'encaminament dinàmic, els nodes envien missatges de control per la xarxa. Aquest missatges consumeixen part de l'ample de banda disponible.

Repetirem, utilitzant encaminament dinàmic amb AODV, les dues proves realitzades en els escenaris de topologia en línia. Comparant els nous resultats amb els obtinguts utilitzant encaminament estàtic, podrem veure quin és el retard afegit per el descobriment de ruta, i quin és l'ample de banda consumit pels missatges de control del protocol d'encaminament.

En el cas de l'encaminament dinàmic podem configurar també escenaris on un origen té més d'un camí possible per arribar un destí. En aquest cas, veurem quines decisions pren el protocol per trobar la que ell considera ruta òptima per a un destí, i com busca una nova ruta quan la utilitzada és invalidada, per exemple, pel trencament d'un dels enllaços. Un altre cop haurem d'utilitzar *iptables* per emular l'escenari virtual desitjat.

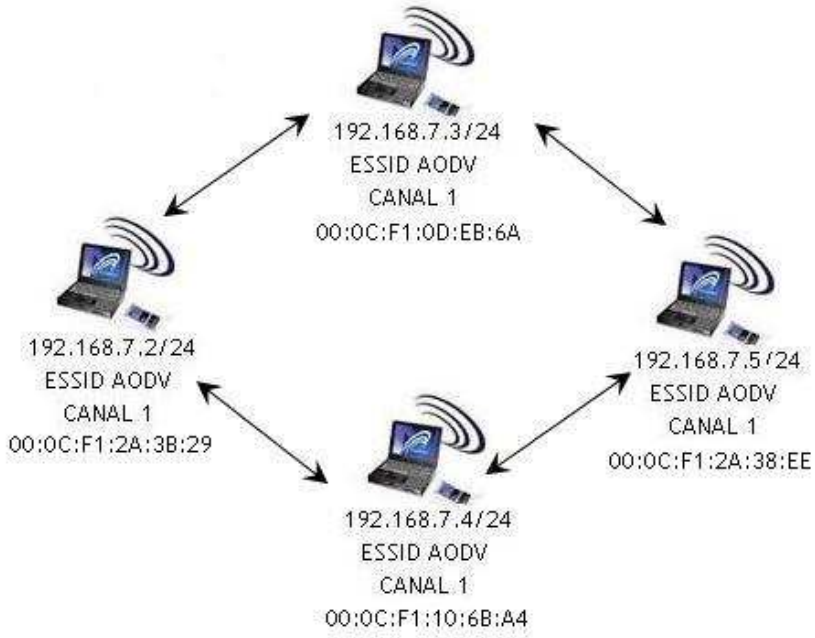


Fig. 5.3 Escenari amb 4 nodes, 2 salts i 2 camins.

Aquestes són les comandes a executar des d'un terminal:

PC 192.168.7.2
iptables -t filter -A INPUT -m mac --mac-source 00:0C:F1:2A:38:EE -j DROP
PC 192.168.7.3
iptables -t filter -A INPUT -m mac --mac-source 00:0C:F1:10:6B:A4 -j DROP
PC 192.168.7.4
iptables -t filter -A INPUT -m mac --mac-source 00:0C:F1:0D:EB:6A -j DROP
PC 192.168.7.5
iptables -t filter -A INPUT -m mac --mac-source 00:0C:F1:2A:3B:29 -j DROP

La decisió de treballar amb un tipus o altre d'escenari vindrà justificada per la característica del protocol que vulguem analitzar.

5.3 Escenaris dinàmics amb FreeBSD

Com ja s'ha esmentat abans, ATCP hauria estat implementat en un PC de sobretaula, amb el sistema operatiu FreeBSD 4.2. De totes maneres, farem servir aquest sistema operatiu per analitzar el rendiment de la implementació de TCP de FreeBSD dins la xarxa ad-hoc, així que en descriurem el seu muntatge. No existeix cap implementació d'AODV per aquesta versió de FreeBSD, així que enllaçarem aquest ordinador amb la xarxa ad-hoc pura mitjançant un cable Ethernet creuat. Connectarem el cable al primer dels portàtils (192.168.6/7.2). Aquest haurà de realitzar funcions de *gateway* i de NAT, ja que si no els paquets arribaran a aquest equip, però no a la resta.

Per a que aquest portàtil realitzi la funció de *gateway* entre les dues xarxes, podem seguir la configuració comentada a l'apartat 5.2.1, on modificant el registre de Linux hem aconseguit que l'ordinador sigui capaç de reenviar paquets. Si fem servir, però, el dimoni AODV d'Uppsala, això no serà necessari, ja que el registre de Linux és modificat pel propi programa i l'opció de *forwarding* és habilitada automàticament (veure annex H).

Si intentem establir una comunicació amb la configuració realitzada fins ara, el portàtil que actua de *gateway* serà capaç d'enviar els paquets de la màquina FreeBSD a la resta dels portàtils de la xarxa ad-hoc, però els portàtils no seran capaços de respondre a l'origen. Els ordinadors de la xarxa ad-hoc reben la informació d'una xarxa que ells no coneixen (192.168.6.x), i no saben com respondre a l'origen. En el cas d'una comunicació unidireccional dirigida de la màquina FreeBSD a un dels portàtils, això no és cap problema, però si aquesta és bidireccional, com en el cas del TCP, la connexió mai no es podrà establir. No serveix definir rutes estàtiques des d'aquest portàtils cap a la màquina FreeBSD, indicant que el *next-hop* d'aquesta informació és el portàtil que fa de *gateway*. Si ho fem, els paquets arribaran a aquest, però ell no sabrà com interpretar que la informació va realment dirigida a la màquina FreeBSD. Per solucionar aquest problema, el portàtil que actua com a *gateway* haurà d'emmascarar primer els paquets provinents de la xarxa 192.168.6.x.

Si habilitem aquesta funcionalitat, coneguda com NAT (*Network Address Translation* [19]), el portàtil que rep informació de la màquina FreeBSD modifica el camp d'origen dels paquets que componen la comunicació, introduint-hi la seva pròpia direcció IP. Una vegada fet això, reenvia la informació cap a la xarxa ad-hoc. Els dispositius d'aquesta xarxa sí que coneixen ara l'origen de la comunicacióI poden respondre en el cas de que això sigui necessari. Els paquets dirigits cap a l'origen són enviats al ordinador que ha emmascarat la informació. Ell sap quina és i com ha de reenviar la informació de tornada cap a la màquina FreeBSD. Per a que el portàtil pugui realitzar aquesta funció, haurem de fer servir un altre cop *iptables*. Obrirem un terminal i executarem la següent línia:

```
# iptables -t nat -A POSTROUTING -j MASQUERADE
```

Tot i així, encara no aconseguirem els resultats desitjats; els paquets són emmascarats, però AODV en el *gateway* només realitzarà descobriments per al paquets generats localment, no pels provinents de la màquina FreeBSD. Per aquesta raó, deixarem un procés ping entre el node *gateway* i el destí. Així, una vegada l'AODV en el node *gateway* trobi la ruta per on enviar la informació generada localment, podrem utilitzar-la per encaminar també la informació provinent de la màquina FreeBSD. Les proves que realitzarem en aquest escenari no tenen en compte l'ample de banda utilitzat, per tant, l'efecte d'aquest tràfic de fons en els resultats és nul. Generarem el ping cada 0,2 s, ja que si el deixem establert a 1s, quan hi hagi un trencament, AODV només podrà reaccionar amb 1 s de resolució (recordem que només actua a la informació generada localment), obtenint resultats no desitjats. Per establir aquest ping, executarem des d'un terminal en el *gateway*:

```
# ping 192.168.7.5 -i 0.2
```

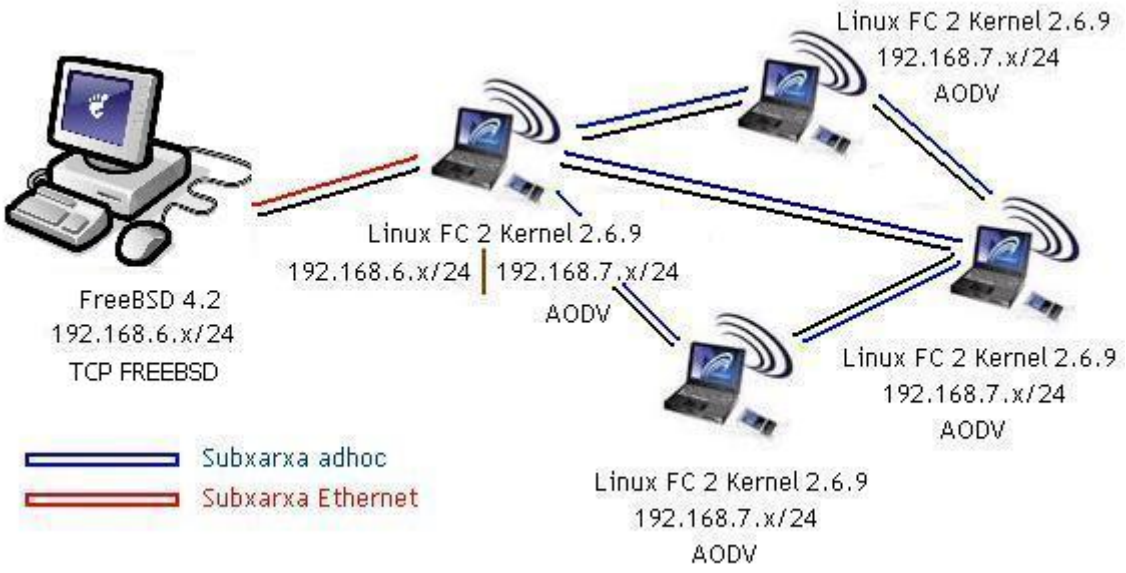


Fig. 5.4 Escenari anterior interactuant amb FreeBSD

CAPÍTOL 6 DEFINICIÓ DE PROVES I ANÀLISI DELS RESULTATS

En aquest capítol exposarem els detalls dels escenaris anteriors i analitzarem els resultats de les proves que s'han realitzat. Coneixerem les característiques dels nostres escenaris, tals com retards i ample de banda disponible, els gaps de connectivitat, el consum de la bateria, i l'efecte de la configuració de l'AODV en tots ells. En la primera part del capítol presentarem els valors obtinguts utilitzant rutes estàtiques, per després comparar aquests resultats amb els obtinguts utilitzant rutes dinàmiques i el protocol AODV. Finalment, estudiarem la interacció del nivell de xarxa amb el nivell de transport, veient com es degrada el funcionament del TCP en xarxes ad-hoc.

6.1 Resultats obtinguts amb rutes estàtiques

En aquest apartat utilitzarem els escenaris estàtics presentats en el capítol anterior. Primer estudiarem el retard dels paquets en funció del nombre de nodes, utilitzant per a les proves un escenari amb tots els equips disposats en línia. Pel fet d'utilitzar rutes estàtiques, definides manualment, els paquets no sofreixen cap retard addicional a l'hora de ser enviats a la xarxa. A més, els nodes no envien cap tipus de missatge de control per tal de mantenir les rutes. D'aquesta manera, també podrem estudiar l'ample de banda disponible amb 802.11b, en funció del nombre de nodes que comparteixen el medi.

6.1.1 Retard del paquets

Per estudiar el retard d'anada i tornada dels paquets en funció del nombre de salts, utilitzarem un escenari amb els nodes disposats en línia, tal i com hem explicat en el apartat 5.1. Recordem que hem d'emular les cobertures desitjades amb *iptables,* habilitant el reenviament de paquets i deshabilitant els missatges *ICMP redirect.*

Amb la comanda ping[20], podem veure quin és el retard d'anada i tornada per cadascun dels paquets enviats. Primerament estudiarem aquests valors amb un sol salt, i anirem més afegint nodes, fins a un màxim de 4 salts. Aquestes són les comandes que hem d'executar des d'un terminal del portàtil 192.168.7.2:

ping 192.168.7.3 -c 10 (per al cas d'un salt)
ping 192.168.7.4 -c 10 (per al cas de 2 salts)
ping 192.168.7.5 -c 10 (per al cas de 3 salts)
ping 192.168.7.6 -c 10 (per al cas de 4 salts)

[20] Per a més informació sobre la comanda, executar *man ping* des d'un terminal.

Quan els nodes no han establert cap comunicació prèviament, no coneixen la relació MAC-IP dels veïns de la xarxa ad-hoc. Si no es coneix aquesta relació per a un destí determinat, l'origen no podrà dirigir-se a aquest, ja que a nivell d'enllaç no sabrà a on enviar la informació. Per solucionar aquest problema, els nodes fan ús del protocol ARP (*Address Resolution Protocol* [20]). Això introdueix un altre retard addicional a l'hora d'establir les rutes. Per evitar això, a les proves s'ha realitzat assignació estàtica de parelles MAC-IP. Trobareu més informació sobre aquest funcionament a l'annex H.

Hem realitzat 15 tests per a cadascuna de les situacions, en funció del nombre de salts que disposem. Cada test ha consistit en una sèrie de 10 pings al destí, en els que hem pogut observar el valor del retard d'anada i tornada que han sofert cadascun d'aquest paquets. Aquest temps també es coneix com RTT (*Round Trip Time*). A la taula 6.1, es presenten els resultats obtinguts:

Taula 6.1 Resultats obtinguts mesurant el retard dels pings

	1 salt	2 salts	3 salts	4 salts
RTT mínim (ms)	1,6	3,15	4,8	6,74
RTT màxim (ms)	3,48	6,42	8,88	10,30
Mitja RTT (ms)	1,94	4,05	5,94	7,95
Desviació estàndard (ms)	0,12	0,23	0,19	0,19

El creixement és lineal segons el nombre de salts que afegim a l'escenari. Podem veure, doncs, com el RTT es duplica amb cada node afegit a la xarxa.

6.1.2 Ample de banda disponible

Per caracteritzar l'ample de banda disponible, farem servir exactament el mateix escenari que a l'apartat anterior. En aquest cas, però, utilitzarem l'eina *iperf* [21]. *Iperf* és una aplicació amb arquitectura client-servidor, en la que el client envia un flux de dades UPD o TCP al servidor durant un període de temps determinat. Un cop finalitzada aquesta transmissió, el receptor de la informació (servidor) envia a l'origen un resum de l'ample de banda mesurat. En el nostre cas, serà el node 192.168.7.2 el que sempre actuï de client. El servidor serà sempre el node destí, i com abans, repetirem les proves per a un número màxim de 4 salts. Per a cadascun dels escenaris, transmetrem 10 fluxos UDP i 10 fluxos TCP durant un període de 30 segons. Els fluxos UDP inserits seran de 6,5 Mbps, saturant així l'ample de banda disponible en el nostre canal de transmissió. Les comandes a executar en cadascun dels portàtils són les següents:

UDP Client (192.168.7.2): # iperf -c *adreçaIPdestí* -u -b 6.5M -t 30
Servidor (destí): # iperf -s -u

TCP Client (192.168.7.2): # iperf -c *adreçaIPdestí* -t 30
Servidor (destí): # iperf -s

Emulant les cobertures desitjades amb *iptables*, aconseguim que un node només es pugui comunicar amb els nodes adjacents, però l'ample de banda del que disposem disminueix en funció del nombre de dispositius que introduïm a la xarxa ad-hoc. Els nodes intermedis comparteixen el medi, i actuen no només com a receptors, sinó també com a fonts d'informació, ja que reenvien els paquets de l'origen. Ens trobem aleshores que per a escenaris de N salts, tenim N-1 fonts d'informació, que han de compartir la capacitat del canal.

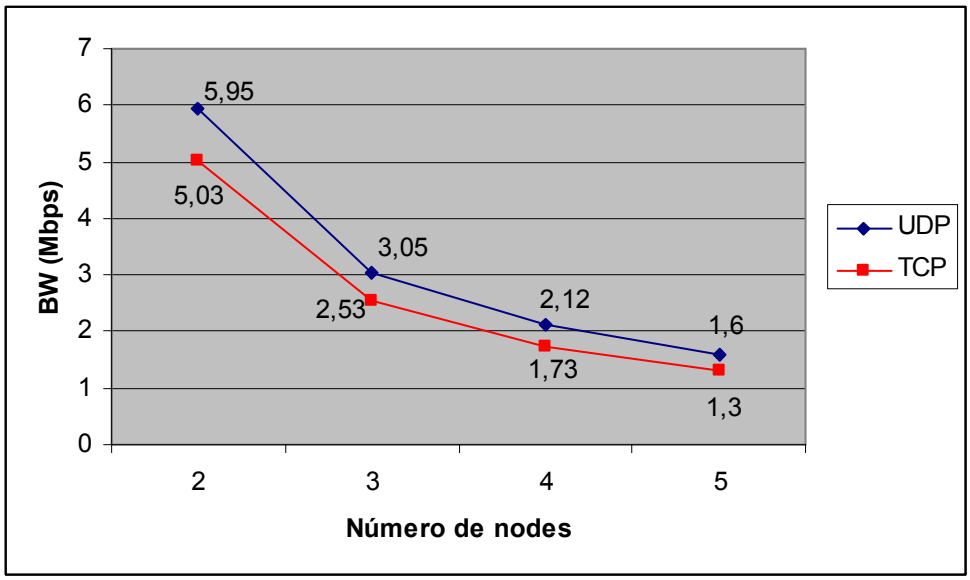


Fig. 6.1 BW disponible en funció del número de nodes

Podem veure a la figura 6.1 com l'ample de banda disponible disminueix amb cada node afegit a la xarxa. Amb 3, 4 i 5 nodes disposem, respectivament, del 50%, 33% i 25 % de l'ample de banda que disposàvem amb només 2 nodes, confirmant-se així la idea de les N-1 fonts d'informació. Al augmentar el número de salts, augmentem el número de fonts que transmeten informació, i això provoca que la capacitat del canal extrem a extrem es vegi disminuïda.

L'ample de banda disponible utilitzant TCP és menor que si utilitzem UDP. Degut als reconeixements de TCP, existeix comunicació en tots dos sentits, al contrari que amb UDP. A més, la capçalera TCP és major que la de UDP. Tot això fa disminuir lleugerament l'ample de banda disponible extrem a extrem.

6.2 Resultats obtinguts amb AODV

6.2.1 Retard afegit per AODV

Per veure quina és la latència afegida per AODV a causa del descobriment de rutes, farem servir un escenari en línia, com als apartats anteriors, però ara les rutes seran descobertes pel protocol d'encaminament.

La configuració de la xarxa és exactament igual que abans, però no assignarem rutes estàtiques. En el seu lloc, configurarem el dimoni AODV en tots els nodes, de manera que aquest serà l'encarregat de trobar els camins per als possibles destins.

Per a aquesta prova, assignarem a tots els paràmetres d'AODV els valors per defecte del RFC, però deshabilitarem l'ús de l'*Expanding Ring Search*, per a que sigui el primer RREQ enviat per l'origen el que arribi al destí. Utilitzant l'*Expanding Ring Search*, el primer dels RREQs només arriba als nodes situats a una distància igual o menor que TTL_START (al RFC, TTL_START=2). En el nostre cas, la xarxa pot arribar a tenir 4 salts, i en aquest cas, l'AODV es veuria forçat a reenviar un altre RREQ per a que aquest arribés al destí. Per habilitar el funcionament de AODV, sense que actuï l'*Expanding Ring Search*, executarem en un terminal a tots els nodes:

```
# aodvd -o --no-expanding-ring
```

Un altre cop assignarem estàticament les parelles MAC-IP dels nodes i realitzarem els 15 tests de 10 pings cadascun. Respecte els valors obtinguts en els escenaris amb encaminament estàtic, només observarem diferència en el valor del primer RTT, ja que aquest és l'únic afectat pel descobriment de la ruta cap al destí. Per a cadascun dels tests, apagarem i tornarem a arrencar el dimoni AODV, ja que així ens assegurarem que cap dels nodes guarda (encara que sigui invalidada) informació de rutes utilitzades anteriorment.

A la següent figura podem veure quina és la mitja del retard obtingut pel primer paquet, que inclou el descobriment, i la mitja del retard dels paquets que no inclouen aquest descobriment. Podem veure també el valor del RTT augmenta linealment en funció dels salts que conté la nostra xarxa.

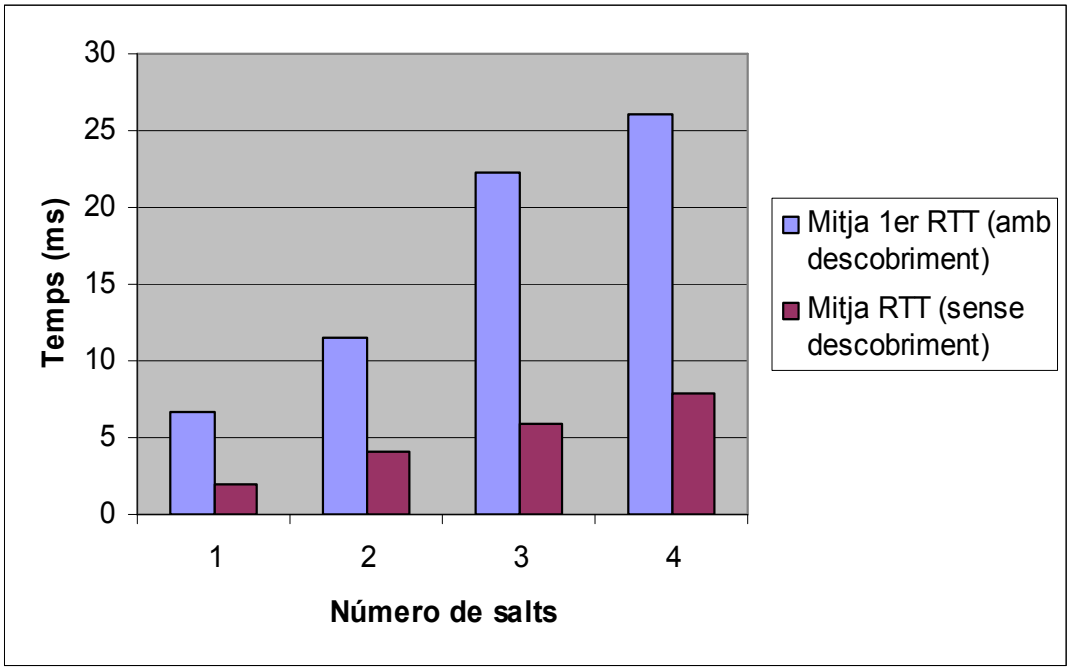


Fig. 6.2 Comparativa RTTs, amb descobriment de ruta i sense descobriment.

Es pot veure que el RTT que inclou el descobriment és aproximadament el doble del RTT que no l'inclou. Podem pensar que, a efectes de temps de propagació, el RREQ és com un ping_*request* i el RREP com un ping_*reply*. Així, la seqüència RREQ + RREP + ping_*request* + ping_*reply* equivaldria temporalment a ping_*request* + ping_*reply* + ping_*request* + ping_*reply*. Però aquesta aproximació és molt distant de la realitat, ja que els nodes triguen un cert temps a processar i reenviar els paquets AODV. Per aquesta raó, observem que com més nodes existeixen a la xarxa, menys es compleix aquesta relació, ja que més nodes són els que processen els paquets AODV, i per tant, més gran és el valor que obtenim pel primer RTT. Es veurà una estimació d'aquest temps de procés dels paquets AODV a l'annex H. Recomanem encaridament la lectura d'aquest annex, ja que es comprova el correcte funcionament d'algunes de les característiques de l'AODV-UU, i s'exposen alguns dels errors i problemes trobats a la implementació.

6.2.2 Ample de banda consumit per AODV

Tant l'escenari com la configuració de l'AODV són els mateixos que l'utilitzat als apartats anteriors. Utilitzarem l'escenari amb els nodes disposats en línia, però amb el dimoni AODV actiu. En aquest cas, ens interessa veure quin és l'ample de banda consumit pels missatges AODV. Aquests missatges són enviats pels nodes mentre aquests mantenen alguna ruta activa. Depenent de la configuració utilitzada, els nodes enviaran missatges de *hello* a intervals de temps més o menys grans, i per tant, disposarem de més o menys d'ample de banda disponible. El que hem de fer és jugar amb el paràmetre HELLO_INTERVAL, que és el que indica cada quan ha d'advertir la seva presència als seus veïns un node que té una ruta activa.

Hem de tenir en compte que com més petit sigui aquest interval, més gran serà la quantitat de tràfic de control enviada a la xarxa, disposant de menys ample de banda per transmetre la informació útil. En canvi, els nodes podran reaccionar abans davant de trencaments, ja que comprovaran més sovint quin és l'estat dels seus veïns. Per a aquests escenaris, hem trobat una sèrie de problemes en el funcionament de AODV quan transmetem fluxos UDP, que es veuen accentuats com més nodes té l'escenari i més petit és el valor de la variable HELLO_INTERVAL.

Primer de tot, s'ha comprovat que per a valors de HELLO_INTERVAL inferiors a 200 ms, els nodes no són capaços de generar els missatges de *hello* exactament quan toca. Per exemple, quan es configura el valor a 50 ms, els nodes envien els missatges de *hello* cada 80 ms, aproximadament. També hem vist en les proves de RTT que els nodes triguen un cert temps a processar els paquets AODV. A més a més d'aquest temps de procés, en aquesta prova estarem saturant el medi, transmetent un flux UDP amb una taxa d'informació superior a la disponible en el canal. Tampoc sabem ben bé quina és la prioritat que estem donant als paquets de control AODV respecte els de l'aplicació *iperf*. El resultat que podem veure és que per a un HELLO_INTERVAL molt petit, succeeix sovint que els nodes no generen els missatges de *hello* quan pertoca, o fins i tot no el generen en algun dels intervals. Això provoca que els nodes no detectin correctament la presència dels seus veïns, i els enllaços es trenquen.

Aquest fenomen es veu més accentuat com més siguin els nodes presents a la xarxa ad-hoc, ja que disminueix l'ample de banda disponible en el canal i més nodes són els que processen els paquets. En el annex H es farà un estudi més acurat d'aquests problemes, i veurem de quina manera es pot millorar aquest rendiment.

A les següents figures podem veure els resultats obtinguts. Hem utilitzat com abans l'aplicació *iperf* per generar el tràfic desitjat. S'han realitzar 10 proves de 30 segons, transferint un flux UDP a 6.5 Mbps, i 10 proves més amb un flux TCP. En aquest cas, però, farem un estudi detallat de cadascun dels escenaris, per al diferent número de nodes que participen de la comunicació. Veurem com amb escenaris de 3 i 4 salts, existeixen els problemes esmentats anteriorment quan utilitzem valors petits per a la variable HELLO_INTERVAL, mentre que amb els escenaris de 1 i 2 salts, els resultats són els esperats. A la figura 6.3 podem veure els resultats per l'escenari de 1 salt.

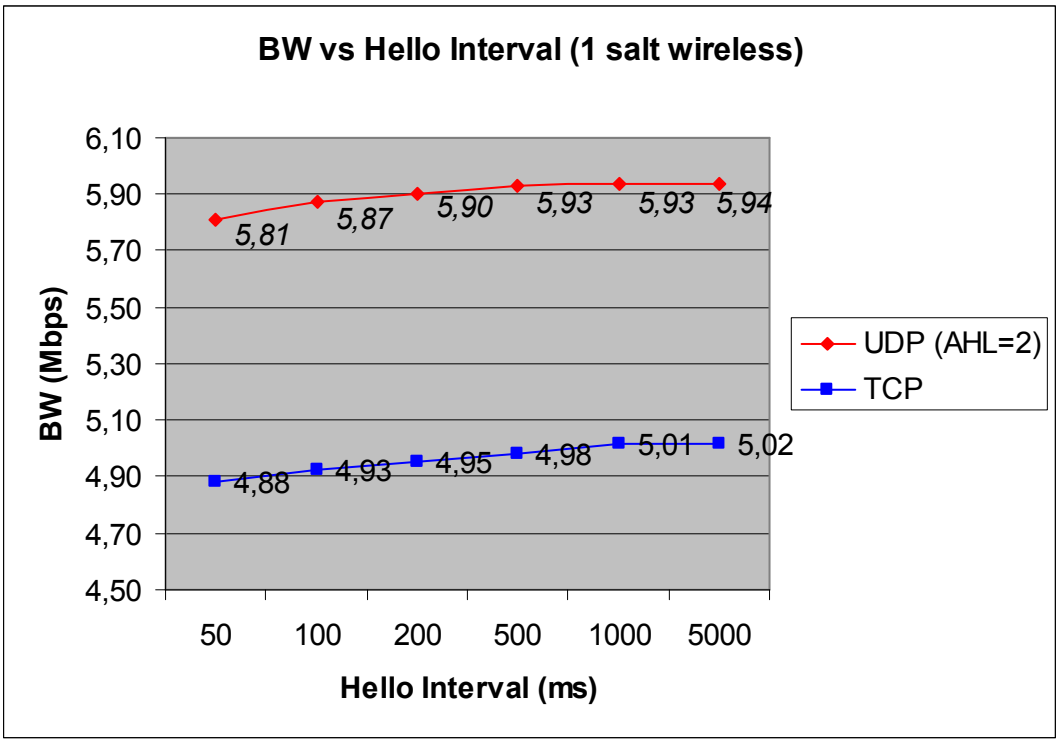


Fig. 6.3 Ample de banda disponible amb 1 salt.

Si l'AODV treballa amb el valor més petit possible per a la variable HELLO_INTERVAL (50 ms), l'ample de banda màxim consumit pels missatges de control del protocol d'encaminament és aproximadament del 2,19 %.

A l'altra banda, si AODV treballa amb el valor més gran possible per a la variable HELLO_INTERVAL (5000 ms), l'ample de banda consumit pels missatges de control és pràcticament 0. Obtenim aleshores el mateix ample de banda disponible que treballant amb rutes estàtiques.

Quan la variable HELLO_INTERVAL té un valor de 500 ms, l'ample de banda disponible per a la informació útil és pràcticament el mateix que si escollim un interval més gran.

Com més baix sigui el valor de la variable HELLO_INTERVAL, el node serà capaç de reaccionar abans als possibles trencaments de l'enllaç, mentre que com més aquest valor, més trigarà en assabentar-se dels possibles trencaments. Els nodes són capaços de detectar els trencaments quan no reben un missatge de *hello* en un període 2*HELLO_INTERVAL d'algun dels seus veïns. Per tant, haurem de trobar el valor d'interval de *hello* que més s'adapti a les nostres necessitats. En entorns on la mobilitat dels nodes és alta, valdrà la pena ajustar el paràmetre HELLO_INTERVAL a valors baixos, mentre que en escenaris on la mobilitat sigui gairebé nul·la, un valor gran de HELLO_INTERVAL ens permetrà transmetre les dades útils per un canal de transmissió amb un ample de banda més gran. Dels resultats extrets d'aquesta prova, sembla ser que un HELLO_INTERVAL de 200 ms o 500ms ens aportarà el millor de tots dos extrems.

Veient que és el que passa en els altres escenaris, haurem de trobar el punt òptim entre rendiment (ample de banda disponible el més elevat possible) i fiabilitat (els nodes no han de trigar gaire en assabentar-se d'un enllaç trencat). A la figura 6.4 podem veure els resultats obtinguts amb un escenari en línia amb 3 nodes, i per tant, 2 salts.

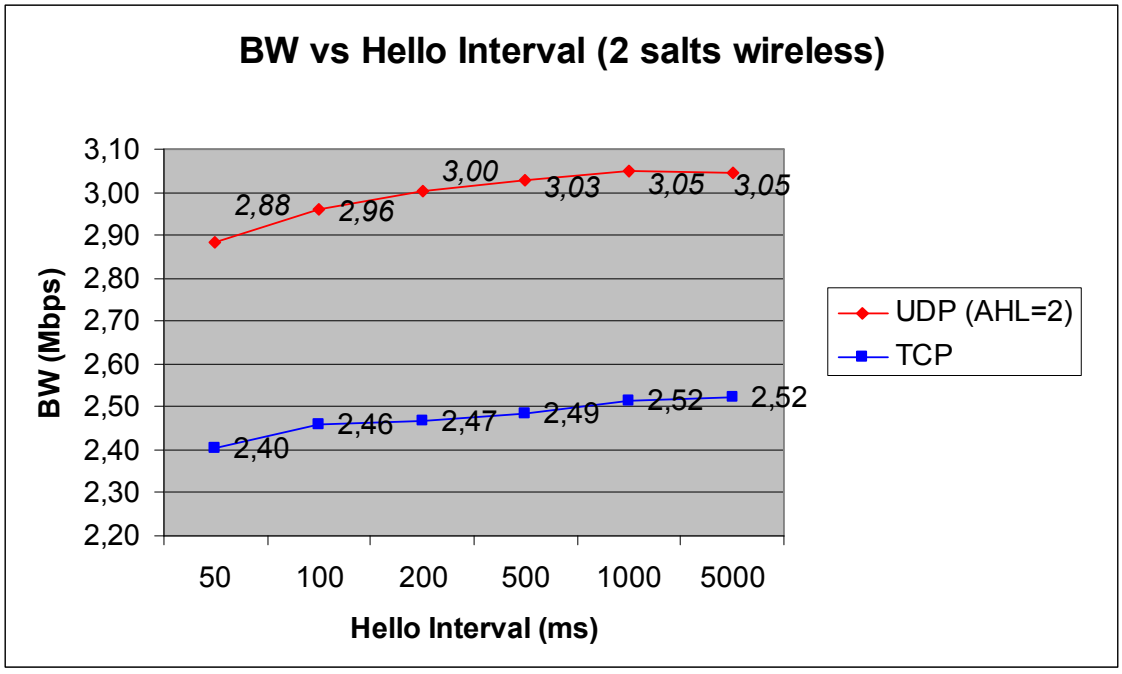


Fig. 6.4 Ample de banda disponible amb 2 salts.

En aquest cas, podem veure com es redueix l'ample de banda disponible aproximadament a la meitat. Ara tenim dos fonts que transmeten informació compartint el mateix medi. Per tant, les dues fonts veuran reduïda la seva capacitat de transmissió aproximadament a la meitat, essent aquesta la

capacitat d'ample de banda total que veurem extrem a extrem. Un altre cop, sembla que el punt òptim per disposar de gairebé tot l'ample de banda ofert per el canal de transmissió, sense renunciar a una ràpida detecció dels trencaments en els enllaços, l'aconseguirem assignant al paràmetre HELLO_INTERVAL un valor de 200 o 500 ms. L'ample de banda consumit en aquest cas per l'AODV és del 5,07 % com a màxim. Al haver-hi més nodes, existeix més tràfic AODV, i aquest guanya presència davant el tràfic generat per *iperf*.

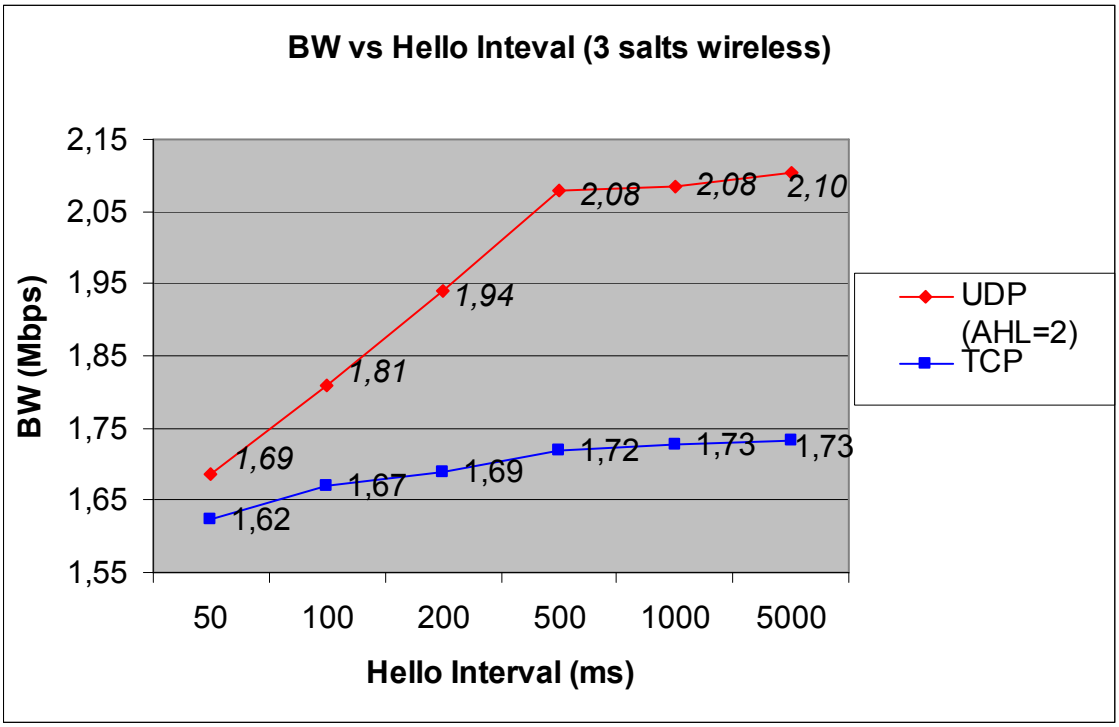


Fig. 6.5 Ample de banda disponible amb 3 salts.

A la figura 6.5 podem veure que en l'escenari de 3 salts ja tenim problemes per a valors de HELLO_INTERVAL inferiors o igual a 200 ms. Ja hem comentat que els nodes no generen els missatges de *hello* exactament segons el que definim a la variable HELLO_INTERVAL, quan aquesta té assignat un valor inferior als 200 ms. Aquest problema no s'aprecia si el escenari té pocs nodes o el tràfic a la xarxa no es gaire elevat, però en aquest cas, amb 3 nodes i saturant el canal amb un flux UDP superior a la capacitat del canal, comencem a veure com es degrada sèriament l'ample de banda disponible utilitzant els intervals de *hello* assenyalats.

Quan un dels nodes no genera els missatges de *hello* segons l'interval especificat, els nodes veïns pensaran que aquest han desaparegut de la xarxa. Sembla ser que els missatges de control AODV no tenen prioritat al ser enviats, respecte els generats per l'aplicació *iperf.* Aquesta situació s'agreuja quan, a més, la taxa UDP de sortida generada per *iperf* és tan elevada. En les condicions esmentades, succeeix sovint que transcorren més de 2*HELLO_INTERVAL ms entre l'enviament de dos missatges de *hello*

consecutius per part d'un dels nodes. Quan això succeeix, els nodes veïns es creuen que l'enllaç ha caigut. Si el node que detecta el trencament és l'origen farà un nou descobriment, mentre que si es un dels nodes intermedis enviarà el RERR als precursors. El resultat és que molts dels paquets enviats per l'origen no arriben al destí, obtenint un ample de banda efectiu més baix que el que tocaria pel que hem vist als escenaris anteriors. En aquest escenari, l'ample de banda màxim consumit per AODV és del 6,36%.

En el cas de TCP, aquesta congestió no existeix, ja que és el propi protocol de transport s'encarrega de que això no succeeixi. Aleshores, podem observar un comportament acord amb el que s'ha vist en els escenaris anteriors. Els trencaments en els enllaços no existeixen i l'ample de banda disponible és el correcte.

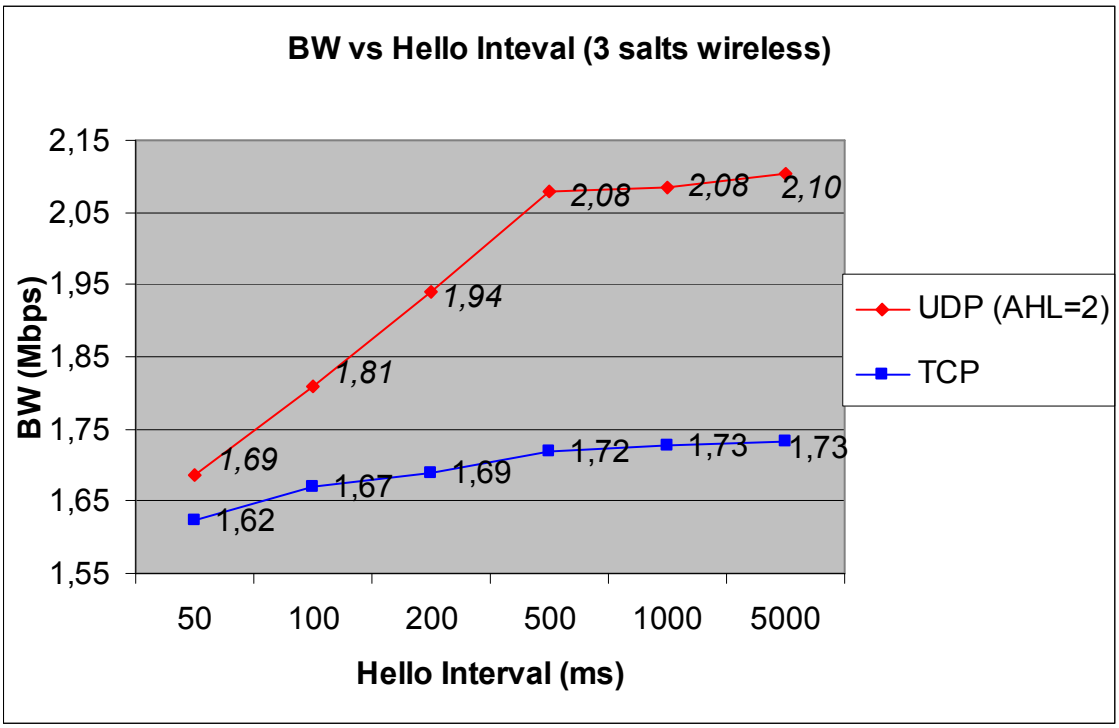


Fig. 6.6 Ample de banda disponible amb 4 salts.

A la figura 6.6 veiem els resultats per l'escenari de 4 salts. El problema de la congestió (i els missatges de HELLO no enviats quan toca) es veu encara més agreujat. Amb un valor de 1000 ms, el problema dels trencaments segueix existint, encara que no és tant sovint com per a valors baixos del paràmetre HELLO_INTERVAL. En els resultats obtinguts en els altres escenaris, podem veure que quan HELLO_INTERVAL val 1000 ms, l'ample de banda disponible a UDP es pràcticament el mateix que quan s'han utilitzat les rutes estàtiques. En aquest cas, l'ample de banda és lleugerament inferior, degut als trencaments en els enllaços. Només per al valor de HELLO_INTERVAL igual a 5000 ms, els efectes d'aquests trencaments són negligibles.

D'aquest dos últims escenaris no en podem treure grans conclusions. En aquest cas, no podem fer un anàlisi correcte del que succeeix per sota del valor 200 ms de HELLO_INTERVAL quan transmetem fluxos UDP. Veient el que succeeix amb TCP, sembla definitivament que utilitzar un valor de 200 o 500 ms ens aportarà rapidesa en les deteccions dels trencaments i un baix consum d'ample de banda. En aquest cas, l'ample de banda màxim consumit per AODV és del 7,5%.

6.2.3 Gaps de connectivitat

És en aquest apartat on veurem la reacció del protocol AODV davant el trencament d'enllaços. Per posar a prova el protocol, farem ús dels escenari presentats als apartats 5.2 i 5.3. Es defineix com a *gap* el temps durant el qual el destí no rep paquets provinents d'una comunicació activa, per culpa del temps que triga el protocol d'encaminament en trobar una nova ruta, davant un trencament d'un enllaç. Farem un estudi de quin és aquest temps, i com afecta a les comunicacions a nivell de transport.

En aquest banc de proves, transmetrem fluxos UDP i TCP al llarg dels escenaris descrits a 5.2 i 5.3 amb l'ús de l'eina *iperf*. En aquests escenaris, l'origen té dos camins per arribar al destí. Trencarem l'enllaç actiu i veurem quant de temps el destí no rep les dades enviades per l'origen. Aquest temps dependrà del valor que haguem ajustat als paràmetres ALLOWED_HELLO_LOSS i HELLO_INTERVAL i del protocol de transport utilitzat.

Per provocar el trencament de l'enllaç actiu, el primer que hem de fer és establir una comunicació entre l'origen (192.168.7.2) i el destí (192.168.7.5), amb l'aplicació *iperf* activa a tots dos nodes. AODV escollirà una ruta, que circularà pel node 192.168.7.3 o pel 192.168.7.4, en funció de quin dels nodes enviï abans la petició de ruta de l'origen al node destí. Per veure quin dels dos nodes és utilitzat en la ruta activa, podem observar amb l'aplicació *Ethereal* [22] de quina MAC provenen els missatges AODV*,* o bé executar des d'un terminal al node origen:

```
# traceroute 192.168.7.5
```

Obtindrem per pantalla quins salts està utilitzant la ruta activa. Una vegada s'ha establert la comunicació, apagarem la interfície sense fils del node intermedi per on circula la ruta activa. Per fer això, podem executar des del terminal:

```
# modprobe –r ipw2100
```

Utilitzant aquest mètode, apagarem la interfície *wireless*, i l'haurem de tornar a encendre posteriorment per a totes les proves que hi realitzarem. Els portàtils utilitzats a les proves, i la majoria dels disponibles al mercat, disposen d'un interruptor que realitza automàticament aquesta funció. Una vegada haguem realitzat aquesta operació, AODV es donarà compte que ha caigut l'enllaç, i s'avisarà a l'origen per tal que iniciï un nou descobriment. Per tornar a habilitar la interfície amb la mateixa configuració, només cal tornar a prémer aquest interruptor.

Utilitzant *Ethereal* al destí, podrem veure quin és el temps que aquest ha estat sense rebre dades noves de l'origen. Aquests temps dependrà del valor dels paràmetres ALLOWED_HELLO_LOSS i HELLO_INTERVAL i del protocol de transport utilitzat. El valor de les dues variables estableix el temps que AODV triga en reaccionar davant el trencament. Aquest estarà comprés en un valor entre HELLO_INTERVAL* (ALLOWED_HELLO_LOSS-1) i HELLO_INTERVAL * ALLOWED_HELLO_LOSS. Això es pot entendre millor a la figura 6.7.

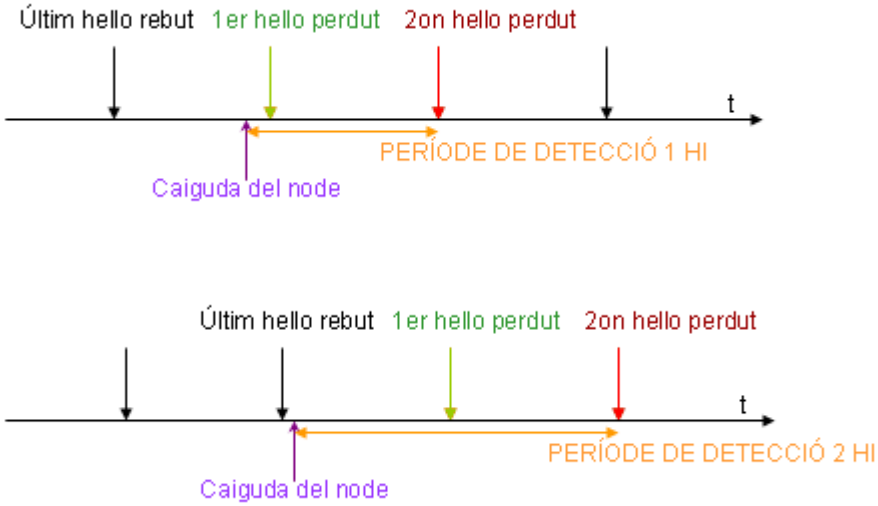


Fig. 6.7 Marge de reacció de AODV segons l'instant de trencament.

Imaginem que es cursa una comunicació que circula a través dels nodes A i B, essent aquest veïns, utilitzant AODV amb els valors per defecte del RFC. Si B cau just en l' instant abans en el que hauria d'enviar un missatge de *hello*, el node A detectarà el trencament de l'enllaç en pràcticament 1 sol HELLO_INTERVAL, ja que durant aquest temps haurà perdut dos missatges de *hello* del seu veí. En canvi, si el node B cau immediatament després d'enviar un missatge de *hello*, el temps que trigarà A per detectar el trencament serà 2*HELLO_INTERVAL.

Hem vist com aquest paràmetres afecten el temps que triga el protocol d'encaminament en detectar el trencament de l'enllaç, però una vegada solucionat el problema, el node destí no té perquè rebre paquets encara, ja que també es depèn del protocol de transport utilitzat. Com veurem més endavant, amb UDP el temps de *gap* és menor que utilitzant TCP, degut a les retransmissions basades en temporitzador que aquest últim realitza quan no rep confirmació per les dades enviades. Aquestes retransmissions depenen de la implementació de TCP utilitzada. En el nostre estudi, farem servir el TCP del kernel de Linux 2.6.9 sobre un escenari com el de la figura 5.2, i el TCP de FreeBSD 4.2 sobre l'escenari de la figura 5.3. Seria molt interessant haver pogut implementar ATCP, ja que és en les situacions que s'han descrit anteriorment on hauríem vist gran part dels seus avantatges.

6.2.3.1 Gaps obtinguts amb UDP

UDP (*User Datagram Protocol*) és un protocol no orientat a connexió, i per tant, no és fiable. No hi ha establiment ni lliurament de la connexió. Les dades s'envien de l'origen al destí, sense cap tipus de control, ja que aquest últim no envia cap tipus d'informació sobre la recepció. Per tant, pot haver-hi desordre en la recepció dels paquets, pèrdues i duplicació.

Quan l'enllaç es trenca, UDP no reacciona de cap manera, ja que no disposa de cap mecanisme de detecció. Per tant, fins que AODV no se n'adona que ha perdut al seu *next-hop* per a la ruta activa, l'origen segueix enviant la informació per la ruta inutilitzada. Totes aquestes dades es perden i no són reenviades. Quan AODV detecta el trencament, immediatament, per al següent paquet a enviar posterior a la detecció, l'origen torna a fer un descobriment de ruta per al destí. S'estableix aleshores la nova ruta pel nou camí, utilitzant l'altre node intermedi. Els paquets tornen a arribar al destí, però la informació que s'ha perdut no pot ser recuperada. El període comprés entre l'últim paquet UDP rebut pel destí a través de la ruta inicial, i el primer paquet UDP rebut a través de la nova ruta, es el que anomenem *gap* de connectivitat d'UDP..

En el cas d'UDP, el temps de detecció del trencament de l'enllaç per part de AODV serà pràcticament igual al temps de *gap* observat al destí. El gap comença tot just quan es trenca l'enllaç, mentre que no finalitza tot just quan aquest trencament és detectat. El gap continua durant el descobriment de la nova ruta per part de l'origen, i fins que el destí no rep de nou un paquet UDP provinent de l'aplicació *iperf*, no es pot donar el *gap* per finalitzat. Podem considerar, doncs, que el temps de gap és:

Temps de gap = temps de detecció de trencament + temps de descobriment de la nova ruta + 0,5 RTT **(6.1)**

A la taula 6.2 podem trobar els diferents temps de gap obtinguts per als diferents valors assignats a la variable HELLO_INTERVAL. ALLOWED_HELLO_LOSS s'ha configurat al valor per defecte del RFC, 2, al igual que la resta de paràmetres configurables del protocol.

Taula 6.2 Gaps obtinguts amb el protocol UDP.

Hello Interval (ms)	**50**	**100**	**200**	**500**	**1000**	**2000**	**5000**
Període de detecció (ms)	[50,100]	[100,200]	[200,400]	[500,1000]	[1000,2000]	[2000,4000]	[5000,10000]
Gap #1 (s)	0,26	0,3	0,32	1,03	1,76	3,33	9,16
Gap #2 (s)	0,28	0,3	0,34	0,86	1,76	3,96	9,65
Gap #3 (s)	0,29	0,28	0,41	0,78	1,35	3,6	6,52
Gap #4 (s)	0,29	0,33	0,3	0,96	1,37	3,9	8,94
Gap #5 (s)	0,28	0,32	0,33	0,93	1,43	3,6	6,16
MITJA GAP UDP (s)	**0,28**	**0,31**	**0,34**	**0,91**	**1,53**	**3,68**	**8,09**

El temps de *gap* és, tal i com hem vist, pràcticament igual al període de detecció del trencament de l'enllaç. Amb la configuració escollida, aquest ha de trobar-se aleatòriament entre els valors HELLO_INTERVAL i 2*HELLO_INTERVAL. Si observem les dades de la gràfica, podem veure que això es compleix amb intervals de *hello* iguals o superiors al 200 ms, però no per als valors de 50 i 100 ms. Si configurem aquest valor per sota dels 200 ms, els intervals són, a la pràctica, més grans que el valor que es configura al fitxer de configuració de AODV-UU, i els nodes no són tan estrictes, parlant en termes de temps, a l'hora de detectar la caiguda dels veïns. Per això, no obtenim els resultats esperats per a valors d'interval de *hello* de 50 i 100 ms.

Durant tot el temps que dura el gap, el destí no rep els paquets UDP generats amb *iperf* a l'origen. Amb l'ajuda d'Ethereal, analitzant la traça de paquets rebuts al destí, podem veure quants són els paquets que s'han perdut mitjançant el camp *Identification* de la capçalera IP. Obtindrem el número de segments perduts restant el valor d'aquest camp en el primer paquet rebut després del restabliment de la ruta al valor d'aquest mateix camp per l'últim paquet UDP abans del trencament. A la figura 6.8 podem veure la traça obtinguda amb *Ethereal* al destí. En ella podem observar el diàleg AODV entre els nodes quan es detecta el trencament. Quan el node origen (192.1697.2) se n'adona que el node intermedi que conduïa la informació al destí (192.168.7.3) ha caigut, inicia un procediment de descobriment enviant un RREQ. Aquest RREQ es rebut pel node intermedi 192.168.7.4, que el reenvia fins que arriba al node destí. El node destí inicia l'establiment de ruta enviant un RREP al node intermedi del que ha rebut el RREQ. En la figura no es veu com el node intermedi reenvia aquest RREP a l'origen perquè a la captura només es mostren els missatges RREQ i RREP de AODV i els datagrames UDP dirigits al destí o missatges *broadcast*. Els missatges de *hello* també han estat filtrats en aquesta captura.

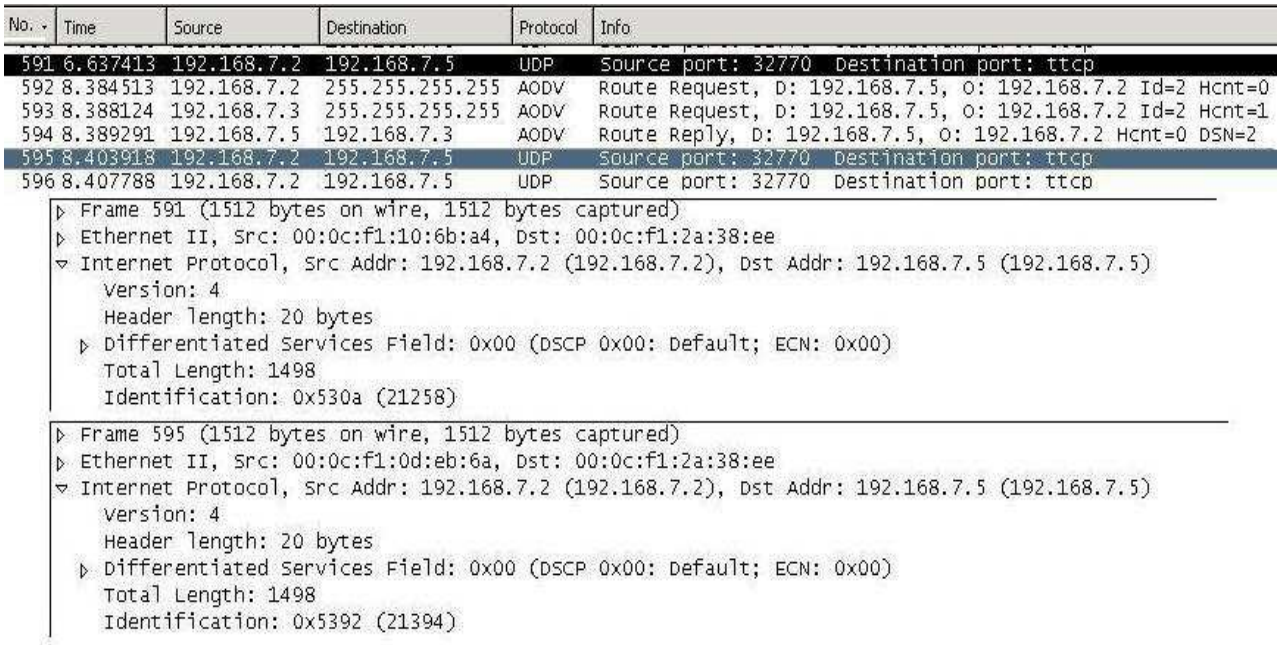


Fig. 6.8 Traça obtinguda amb Ethereal al destí.

Es destaquen dos paquets UDP, l'últim rebut abans del trencament i el primer rebut després de la detecció. Es pot veure també de quina MAC s'han rebut i quant val el valor del camp *Identification* de la capçalera IP. En aquestes proves, el flux UDP injectat amb *iperf* ha estat de 1 Mbps. A cada paquet UDP s'envien 1462 bytes d'informació útil. Podem trobar aquest valor en el camp corresponent[21] dels paquets obtinguts a la traça. Podem saber quants paquets per segon s'envien des de l'origen mitjançant la següent fórmula:

$$1 \text{ Mbps} / (1462 * 8) = 85{,}5 \text{ paquets UDP / segon} \qquad \textbf{(6.2)}$$

La inversa d'aquest valor és la taxa de sortida de paquets a l'origen, i val 11,7 ms. Per verificar que aquest valor és cert, podem veure els instants de temps en el que s'envien paquets, analitzant la traça obtinguda amb *Ethereal* a l'origen. Podem observar una mostra d'aquesta traça a la figura 6.9, i comprovar com, efectivament, es genera un datagrama UDP cada 11,7 ms.

No.	Time	Source	Destination	Protocol
101	0.895444	192.168.7.2	192.168.7.5	UDP
102	0.907192	192.168.7.2	192.168.7.5	UDP
103	0.918969	192.168.7.2	192.168.7.5	UDP
104	0.930702	192.168.7.2	192.168.7.5	UDP
105	0.942461	192.168.7.2	192.168.7.5	UDP
106	0.954246	192.168.7.2	192.168.7.5	UDP
107	0.965989	192.168.7.2	192.168.7.5	UDP
108	0.977741	192.168.7.2	192.168.7.5	UDP
109	0.989531	192.168.7.2	192.168.7.5	UDP
110	1.001287	192.168.7.2	192.168.7.5	UDP

Fig. 6.9 Traça UDP obtinguda amb Ethereal a l'origen.

Coneixent el valor de la taxa de sortida, podem estimar quants paquets s'han perdut, multiplicant aquest valor pel temps que ha durat el *gap*. Compararem aquest valor amb el valor real obtingut de les captures de la figura 6.8. Podem observar els resultats a la taula 6.3. El *gap* mesurat a les captures és més gran que el real, i per aquesta raó, la quantitat de paquets perduts mesurats és menor que la quantitat estimada (a partir d'una mesura amb un cert error).

Taula 6.3 Comparativa paquets perduts teòrics i reals.

Hello Interval (ms)	**50**	**100**	**200**	**500**	**1000**	**2000**	**5000**
Gap UDP (s)	0,28	0,306	0,34	1,054	1,534	3,678	8,086
Gap #1	17	22	26	82	136	270	728
Gap #2	13	20	28	73	131	290	749
Gap #3	19	14	33	77	100	304	545
Gap #4	16	22	23	89	99	321	719
Gap #5	18	21	21	90	108	291	505
Mitja paquets perduts reals	**16,6**	**19,8**	**26,2**	**82,2**	**114,8**	**295,2**	**649,2**
Paquets perduts teòrics	23,8	26	28,9	89,6	130,4	312,8	687,6

[21] Aquest és el camp *Data* de 8 bits, ubicats a la capçalera UDP.

6.2.3.2 Gaps obtinguts amb TCP

TCP (*Transmission Control Protocol*) és un protocol de transport fiable i orientat a connexió. Precisament serà aquest control en la transmissió la causa de que el temps de *gap* sigui més gran que el temps de detecció de trencament de l'enllaç per part de AODV, i per tant, que el *gap* obtingut amb UDP.

Això és degut a les retransmissions que efectua el protocol TCP davant l'absència de reconeixement dels segments enviats. Quan l'origen envia un segment TCP, posa en marxa un temporitzador de retransmissió, que depèn bàsicament del RTT existent entre l'origen i el destí. L'origen segueix enviant segments mentre no s'omple la finestra de transmissió. Si el temporitzador de retransmissió (RTO) expira per a algun dels segments, sense haver rebut l'ACK corresponent, es retransmet el segment pertinent. En el moment en el que es trenca l'enllaç actiu, ni els nous paquets enviats per l'origen es reben al destí, ni els reconeixements del últims paquets reenviats pel node intermedi al destí, just abans del trencament, arriben a l'origen. Quan això succeeix, s'omple la finestra de transmissió a l'origen, que no pot enviar més segments fins que algun sigui reconegut pel destí. Expira aleshores el RTO del primer dels segments no reconeguts i el node origen el retransmet. Si rep resposta per part del destí, pot tornar a enviar dades noves, però si no en rep, ha d'esperar a la propera retransmissió pel paquet. Aquestes retransmissions segueixen un *backoff* exponencial, és a dir, el RTO es duplica amb cada reintent fallit.

Les retransmissions de TCP són la causa de que el temps de *gap* amb TCP sigui major que utilitzant UDP. L'AODV detecta, com abans, el trencament de l'enllaç en un període comprès entre HELLO_INTERVAL i 2*HELLO_INTERVAL, ja que seguim usant els valors per defecte del RFC. Durant aquest temps, però, s'haurà omplert la finestra de transmissió a l'origen i haurà expirat el RTO del primer dels segments TCP de la finestra. Encara que AODV hagi detectat i reparat el trencament, no arribaran segments TCP al destí fins que no expiri el RTO de TCP, ja que l'origen no podrà enviar dades fins aleshores. Depenent del valor de HELLO_INTERVAL configurat, caurem en una o altra d'aquestes retransmissions, a causa del *back-off* exponencial, i el gap serà més o menys llarg. Pot ajudar a entendre el concepte la següent figura.

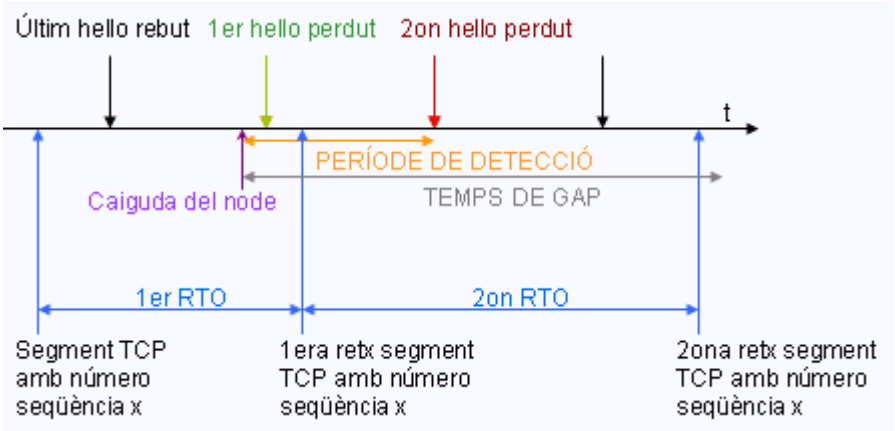


Fig. 6.10 Amb TCP, el temps de gap no és el temps de detecció.

A la figura es veu com, en aquest cas, en el moment de la detecció el node es troba entre la primera i la segona retransmissió d'un segment x. Entre quines dues retransmissions ens trobem depèn directament del període de detecció, i en conseqüència, de l'interval de *hello* escollit.

A la taula 6.4, podem trobar els resultats de gap obtinguts amb TCP Linux per a les diferents valors possibles de la variable HELLO_INTERVAL. A la taula 6.5 es mostren els resultats amb TCP FreeBSD 4.2. Aquests valors s'han obtingut un altre cop a partir de les traces capturades amb *Ethereal*. Degut a la complexitat en l'anàlisi de les traces d'aquestes proves, en detallarem l'anàlisi l'annex H. Veurem que els TCP de Linux i FreeBSD calculen els RTO de manera diferent, i per això, obtenim diferents temps de *gap* segons la implementació utilitzada.

Taula 6.4 Gaps obtinguts amb el protocol TCP Linux.

Hello Interval (ms)	50	100	200	500	1000	2000	5000
Període de detecció (ms)	[50,100]	[100,200]	[200,400]	[500,1000]	[1000,2000]	[2000,4000]	[5000,10000]
MITJA GAP TCP LINUX (s)	**0,47**	**0,55**	**1,26**	**1,26**	**2,91**	**6,38**	**9,08**

Taula 6.5 Gaps obtinguts amb el protocol TCP FreeBSD.

Hello Interval (ms)	50	100	200	500	1000	2000	5000
Període de detecció (ms)	[50,100]	[100,200]	[200,400]	[500,1000]	[1000,2000]	[2000,4000]	[5000,10000]
MITJA GAP TCP FREEBSD (s)	**1,01**	**1,06**	**1,82**	**2,99**	**3,00**	**7,00**	**14,99**

6.2.3.3 Conclusions

Com hem vist, els temps de *gap* obtinguts amb el protocol AODV són aproximadament els esperats. Utilitzant UDP per transportar les dades, els *gaps* equivalen pràcticament al temps de detecció del protocol d'encaminament, mentre que utilitzant TCP, el temps de *gap* és major a causa de les retransmissions temporitzades davant l'absència de reconeixement de dades. Les implementacions de TCP i FreeBSD no calculen de la mateixa manera aquest temporitzadors, i per això originen resultats diferents.

En qualsevol cas, els temps obtinguts pels *gaps* amb AODV són justificables, no com els obtinguts amb el protocol d'encaminament proactiu OLSR, en un treball molt similar a aquest, dut a terme per en David García Robles (veure [23]). Amb OLSR, s'han obtingut gaps per UDP que van dels 7 als 21 s, segons la configuració del protocol utilitzada (segurament per un mal funcionament de

la implementació OLSR de Unik, ja que el seu autor, tampoc podia explicar aquests resultats).

A la següent figura podem observar un resum dels *gaps* obtinguts per a les diferents configuracions d'AODV, en funció del protocol de transport utilitzat.

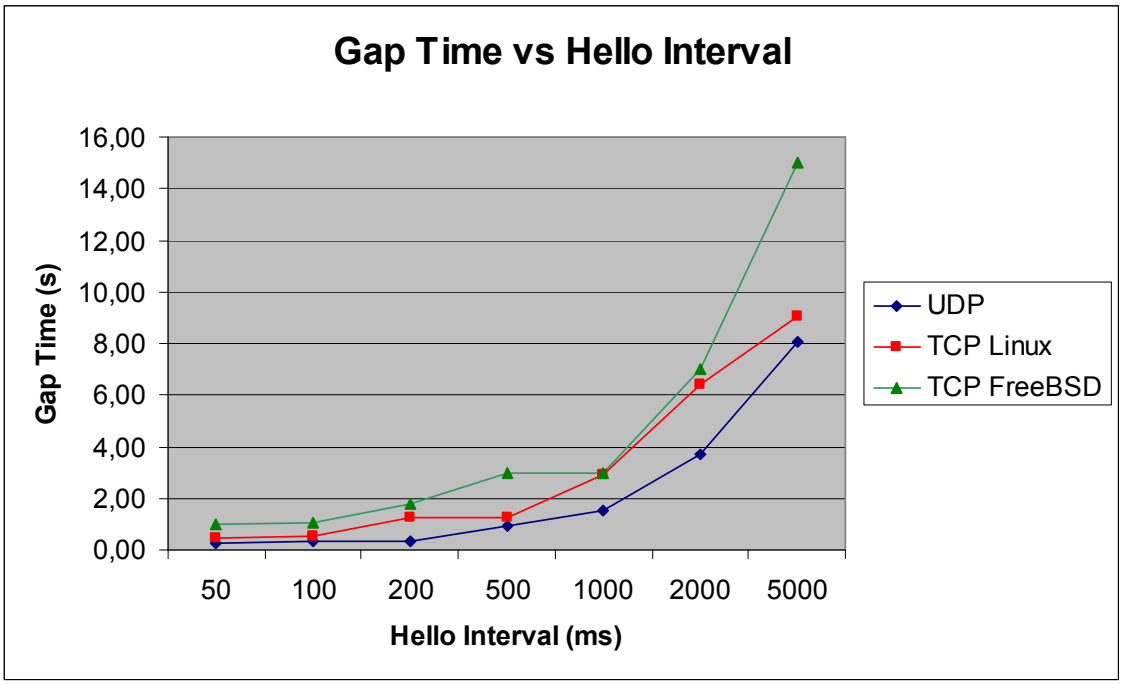


Fig. 6.11 Resum dels *gaps* obtinguts.

6.2.4 Duració de la bateria

La duració de la bateria en els dispositius portàtils mòbils depèn en bon grau de l'ús que es faci de les seves capacitats *wireless*. Per aquesta raó, és de suposar que com més es faci servir aquest medi de transmissió, més s'acurtarà la vida de la bateria. En el nostre cas, això ens pot fer decidir utilitzar un valor determinat pel paràmetre HELLO_INTERVAL de AODV.

Hem de tenir en compte que com més petit sigui aquest valor, abans detectarem la caiguda dels nostres veïns, però s'enviaran més missatges de control a la xarxa i això repercutirà directament a la vida de la bateria dels portàtils. Si el flux de dades a enviar és molt alt, el valor escollit per a la variable HELLO_INTERVAL no repercutirà gaire en la vida de la bateria, mentre que si aquest és baix, sí que pot tenir certa importància, degut a la diferència dels fluxos en tots dos casos.

En el nostre estudi hem optat per transmetre un flux TCP entre els dos extrem d'un escenari en línia amb 3 nodes. La vida de la bateria ha estat mesurada en el node intermedi, ja que és el que més tràfic cursa de tots tres. Els nodes només cursaran la comunicació TCP i la comunicació AODV associada al

manteniment de la ruta. Tots 3 nodes cursaran aproximadament el mateix tràfic AODV, mentre que el node intermedi haurà de rebre i reenviar dos segments TCP en el mateix temps que els nodes extrems reben i envien un de sol cadascun.

La bateria del node intermedi s'ha carregat al màxim, i s'ha desconnectat el portàtil de l'alimentació just en el moment d'iniciar la transmissió del flux TCP. S'ha configurat la pantalla del portàtil per tal que s'apagui al minut d'iniciar aquesta transmissió. S'ha considerat que la bateria s'ha esgotat en el moment que la pantalla es torna a encendre per mostrar-nos l'avís que només queda un 3% de bateria. Quan això succeeix, la comunicació TCP es talla, ja que es reinicia la sessió gràfica de Linux, tancant-se tots els terminals existents, entre els que hi ha inclòs el que executa el dimoni AODV.

A la taula 6.6 es pot veure la duració de la bateria en el node intermedi depenent del valor escollit per al paràmetre HELLO_INTERVAL. Podem veure com la bateria s'esgota abans al escollir un valor petit de interval de *hello*.

Taula 6.6 Duració de la bateria en funció de HELLO_INTERVAL

Hello Interval (ms)	50	100	500	1000	5000
Duració bateria (m)	312	319	339	348	355

El flux TCP és molt major al generat per AODV. Utilitzant un interval de *hello* de 50 ms, la bateria només ens durarà un 12,11% menys que si li assignem el valor 5000 ms.

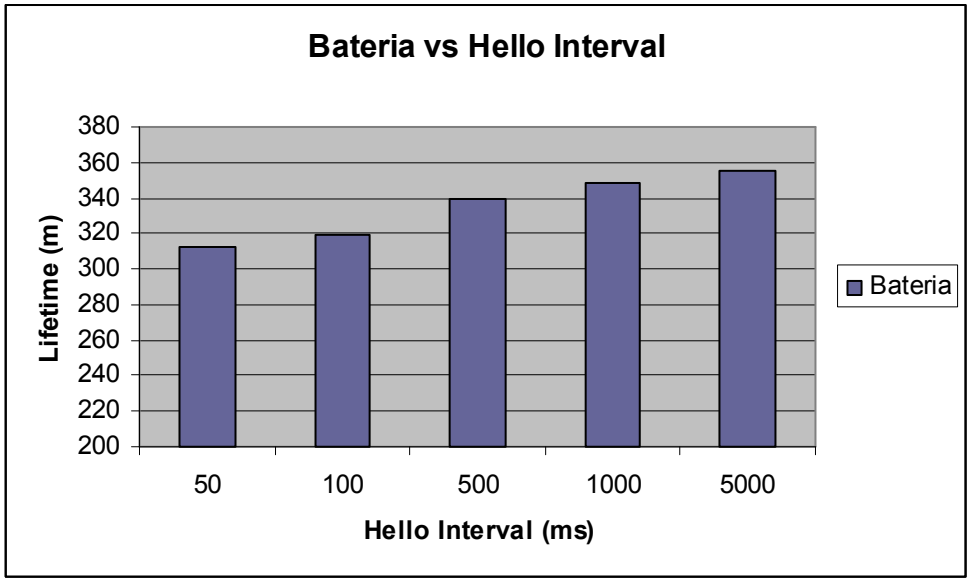


Fig. 6.12 Duració de la bateria segons configuració AODV.

CONCLUSIONS

En aquest treball hem après a configurar una xarxa ad-hoc amb el protocol d'encaminament AODV. Hem vist com funciona aquest protocol, en concret, la implementació AODV-UU, i l'hem analitzat. Les conclusions que podem extreure de les proves realitzades són el prou satisfactòries, quan s'escullen els paràmetres adequats del protocol. La configuració escollida de AODV té efectes en el rendiment de la xarxa ad-hoc.

La latència afegida en el descobriment de les rutes no ha estat del tot satisfactòria quan la xarxa ad-hoc superava un cert nombre de nodes, tot i que la causa d'això ha estat un mal comportament de la implementació AODV-UU.

L'ample de banda consumit pel protocol és prou baix, i en conseqüència, també ho és el consum de bateria. A més, aquest consum és nul quan no hi hagi rutes actives a la xarxa. Configurant el valor d'interval de *hello* a 500 ms, aquest consum és gairebé despreciable, mentre que obtenim una bona resposta en la detecció del trencament dels enllaços actius.

El temps de resposta davant la detecció dels enllaços ve donada per la configuració escollida d'AODV. Aquest temps val com a màxim ALLOWED_HELLO_LOSS*HELLO_INTERVAL. Escollint el valor d'interval de *hello* a 500 ms, sense canviar el valor per defecte de la resta de paràmetres, el temps de detecció mai no serà major a 1s. Configurant el valor d'interval de *hello* a 200 ms, els nodes a vegades no duen a terme la detecció correctament, obtenint a la pràctica temps de detecció similars als obtinguts amb 500 ms. Estarem doncs malgastant recursos escassos en el món ad-hoc, com són l'ample de banda i la bateria.

El temps que triguen els nodes en detectar els trencaments i trobar la nova ruta té conseqüències directes a la capa de transport. El TCP degrada molt el seu funcionament quan les particions a la xarxa són constants, i aquesta és la naturalesa de les MANETs.

Utilitzant UDP per transportar les dades, el temps que el destí no rep dades noves equival pràcticament al temps de detecció del protocol d'encaminament, mentre que utilitzant TCP, el temps de *gap* és major, a causa de les retransmissions temporitzades davant l'absència de reconeixement de dades. El TCP de Linux efectua aquestes retransmissions abans que la implementació realitzada a FreeBSD, per tant, sembla una implementació més apta per a escenaris ad-hoc.

LÍNIES FUTURES

Actualment, AODV no especifica cap tipus de mesura de seguretat. Existeix, però, un *draft* [24] en procés al respecte. Els protocols d'encaminament són objectiu fàcil d'atacs de suplantació d'identitat. A xarxes on els membres no són coneguts, és molt difícil de prevenir aquests atacs. Els missatges AODV tenen que ser protegits mitjançant tècniques d'autenticació, com firmes digitals i resums dels missatges. Per exemple, en un cas d'atac, un missatge RREP podria ser modificat per canviar la ruta del destí, enviant les dades a l'usuari maliciós. De la mateixa manera, amb els missatges RERR es podria impossibilitar la comunicació al llarg d'una xarxa. Tots aquest atacas poden donar inconsistència a les taules d'encaminament dels nodes de la xarxa.

Respecte a la feina realitzada exclusivament en aquest treball, seria molt interessant veure altres implementacions de AODV. AODV-UU treballa en l'espai d'usuari. Això provoca que el temps de procés als nodes sigui majors, en teoria, que si s'hagués utilitzat una implementació que modifiqués el propi kernel, com Kernel AODV. Potser amb aquesta implementació es poden obtenir encara millors resultats en la detecció dels trencaments. També seria interessant provar la interacció entre diferents implementacions del protocol, encara que és difícil obtenir resultats coherents amb la varietat d'implementacions existents i les seves característiques (moltes no compleixen del tot el RFC). D'aquesta manera, es podria analitzar el rendiment del protocol en una xarxa del tot heterogènia. Recordem que també existeix la possibilitat de cross-compilar AODV-UU per a dispositius portàtils com les Zaurus.

Respecte la implementació real de la xarxa, sabem que emular les zones de coberura filtrant per la MAC no és l'escenari ideal, però sense poder tenir control sobre la potència de transmissió de les targetes utilitzades, és senzillament impossible, ja que necessitaríem un enorme espai "blanc" (sense cap tipus de rebot ni interferència). També seria interessant provar altres topologies o veure el funcionament d'aquesta implementació utilitzant la detecció a nivell d'enllaç.

Utilitzant altres implementacions de l'AODV, seria força interessant veure quin és el rendiment del protocol sobre altres tecnologies a nivell d'enllaç, com per exemple, ZigBee (802.15.4)

A la mateixa universitat de Uppsala s'està preparant una nova implementació del protocol reactiu DSR. Vist el bon funcionament d'AODV-UU, és una bona opció començar a testejar el funcionament d'aquesta implementació tan bon punt aparegui.

Finalment no hem pogut analitzar el rendiment del protocol ATCP. Hem comptat amb l'ajuda de Harkirat Singh, un dels coautors de la implementació, per a la instal·lació, però no ha estat possible. Hem intentat esbrinar si algú ha fet servir mai aquesta implementació i no hem rebut resposta. Seria interessant continuar amb aquesta investigació.

IMPLICACIONS MEDIAMBIENTALS

Les implicacions mediambientals de les aplicacions estudiades en aquest treball són bastant positives.

L'ús de xarxes sense fils permet comunicar-nos sense cables. Aquest cables, normalment, estan fabricats amb coure, i aquest comença a ser un recurs escàs a la natura. Per tant, l'ús d'aquestes xarxes permet estalviar costos de fabricació i no malgastar aquest element, a més de no produir un impacte mediambiental del tipus estètic.

L'altra lectura positiva l'obtenim amb les aplicacions possibles gràcies a l'ús de la tecnologia ad-hoc. Podem desplegar immenses xarxes de sensors, de baix cost i gran autonomia de bateria, que ens permeten detectar incendis a boscos, radiacions, o altres tipus de perills mediambientals. També poden ser de gran ajuda en situacions de desastre, tant per restaurar les infraestructures de comunicacions danyades com per ajudar als cossos que intervinguin en aquest tipus d'operacions.

BIBLIOGRAFIA

[1] C. Perkins, E.M.Belding-Royer, S. Das ”Ad hoc On-Demand Distance Vector (AODV) Routing” RFC 3561, 2003

[2] E. Belding-Royer, C. Perkins, “Evolution and future directions of the ad hoc on-demand distance-vector routing protocol”, 2003.

[3] C. Gómez, J. Paradells “Redes ad-hoc: el próximo reto”, 2003

[4] J. Liu, S. Singh “ATCP: TCP for Mobile Ad-Hoc Networks”, 2003

[5] L, Klein-Berndt “A quick guide to AODV routing”, 2003

[6] B, Wiberg “Porting AODV-UU Implementation to ns-2 and Enabling Trace-based Simulation”, 2002

[7] T. Clausen, P.Jacquet “Optimized Link State Routing Protocol (OLSR)” RFC 3626, 2003

[8] P. Sarolahti, A. Kuznetsov “Congestion control in Linux TCP”, 2002

[9] A. Kesselman, Y. Mansour “Optimizing TCP retransmission timeout”, 2004

[10] C. K. Toh, “Ad Hoc Mobile Wireless Networks: protocols and systems”, Prentice-Hall, 2002

[11] P. Misra “Routing Protocols for Ad Hoc Mobile Wireless Networks”, 2000

[12] Com recompilar el kernel de FreeBSD. Web. http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/kernelconfig-building.html

[13] Tutorials de Linux. Web. http://www.linux.org/

[14] Configuració de iptables. Web. http://www.linuxguruz.com/iptables/howto/

REFERÈNCIES

[1] MANET. Grup de treball de l'IETF (*Internet Engineering Task Force*). Pàgina oficial.
http://www.ietf.org/html.charters/manet-charter.html
Discusions generals: manet@ietf.org
Subscripció: mail a manet-request@ietf.org. Al cos, *subscribe manet*
Arxiu: http://www.ietf.org/mail-archive/web/manet/index.html

[2] Bluetooth (802.15.1) i ZigBee (802.15.4). Respectives pàgines oficials.
http://www.bluetooth.com/
http://www.zigbee.org

[3] Protocols MANET proactius.
OLSR, RFC 3626, www.ietf.org/rfc/rfc3626.txt
TBRPF, RFC 3684, http://www.faqs.org/rfcs/rfc3684.html

[4] Protocols MANET reactius.
AODV, RFC 3561, http://www.faqs.org/rfcs/rfc3561.html
DSR, draft http://www-2.cs.cmu.edu/~dmaltz/internet-drafts/draft-ietf-manet-dsr-09.txt

[5] Protocols MANET híbrids.
ZPR, draft http://people.ece.cornell.edu/~haas/wnl/Publications/draft-ietf-manet-zone-brp-02.txt
SHARP, article http://www.cs.cornell.edu/People/egs/615/sharp.pdf

[6] Universitat de Califòrnia (UCLA). Web oficial.
http://www.ucla.edu/
Universitat de Cincinnati (UC). Web oficial.
http://www.uc.edu/

[7] Network Simulator. Simulador de xarxes. Web oficial.
http://www.isi.edu/nsnam/ns/

[8] Pàgina oficial del NIST (*National Institute of Standards and Technologies*).
http://www.nist.gov/

[9] Web oficial de Tiny OS.
http://www.tinyos.net/

[10] Web oficial de la universitat de Uppsala (UU).
http://www.uu.se/

[11] ECN (pàgina del *Center Internet Research*).
http://www.icir.org/floyd/ecn.html

[12] Web oficial del projecte Fedora Core.
http://fedora.redhat.com/

[13] Web oficial de les eines de configuració Wireless Tools.
http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html

[14] Web oficial de FreeBSD.
http://www.freebsd.org

[15] Web oficial dels drivers IPW2100.
http://ipw2100.sourceforge.net/

[16] Webs oficials dels entorns gràfics basats en X11 de Linux.
http://www.gnome.org/
http://www.kde.org/

[17] Web oficial del projecte gcc, compilador de Linux.
http://gcc.gnu.org/

[18] Article sobre l'encriptació WEP.
http://www.wi-fiplanet.com/tutorials/article.php/1368661

[19] NAT, RFC 1631, http://www.faqs.org/rfcs/rfc1631.html

[20] ARP, RFC 826, http://www.faqs.org/rfcs/rfc826.html

[21] Iperf. Eina per mesurar el rendiment de la xarxa. Web oficial del projecte.
http://dast.nlanr.net/Projects/Iperf/

[22] Ethereal. Analitzador de protocols. Pàgina web oficial.
http://www.ethereal.com/

[23] *Realización y análisis de una red móvil ad-hoc con protocolo de encaminamiento OLSR*. García Robles, David .6 de setembre del 2004. TFC EPSC.

[24] Secure AODV. Draft. http://www.cs.ucsb.edu/~ebelding/txt/saodv.txt

[25] Web projecte PACMAN. Implementació ATCP. http://www.cs.pdx.edu/~singh/pacman.html

[26] Web oficial del kernel de Linux
http://www.kernel.org

Escola Politècnica Superior
de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

ANNEXOS

TÍTOL: Avaluació d'una xarxa ad-hoc real amb protocol d'encaminament AODV

AUTOR: Xavi Mantecón Ferrer

DIRECTOR: Carles Gómez Montenegro

CO-DIRECTOR: Marisa Catalán Cid

DATA: 25 de febrer de 2005

A. AODV

A.1. Format dels missatges AODV

- Route Request (RREQ)

Byte 0								Byte 1								Byte 2								Byte 3							
0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
Tipus (1)								J	R	G	D	U	Reservat											Hop Count							
RREQ ID																															
Direcció IP del destí																															
Número de seqüència del destí (DSN)																															
Direcció IP de l'origen																															
Número de seqüència de l'origen (OSN)																															

J *Join flag*; reservat per multicast. Serveix per unir-se a un grup.
R *Repair flag*; reservat per multicast.
G *Gratuitous* RREP *flag*; sol·licita explícitament l'enviament d'un *Gratuitous* RREP al destí del RREQ.
D *Destination only flag*; sol·licita que el RREQ només pugui ser contestat pel destí.
U *Unknown sequence number*; s'activa aquest *flag* quan el DSN del destí és desconegut.

Reservat:
Opcions futures. S'envia com a 0, i és ignorat al destí.

Hop Count:
Número de salts que separen el node que reenvia el RREQ fins l'origen de la petició.

RREQ ID:
Número de seqüència, que juntament amb la IP el node origen, forma un missatge únic.

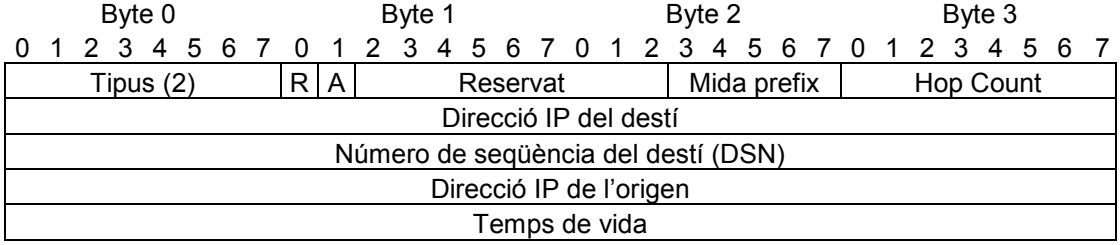
Direcció IP del destí:
Direcció IP del destí pel que es desitja la ruta.

Número de seqüència del destí:
L'últim DSN conegut per al destí per part de l'origen.

Direcció IP de l'origen:
Direcció IP de l'origen que ha sol·licitat la ruta.

Número de seqüència de l'origen:
El número de seqüència actual que ha de ser utilitzat en la ruta inversa cap a l'origen.

- Route Reply (RREP)



R *Repair flag*; reservat per multicast.
A *Acknowledgment required flag*; sol·licita explícitament l'enviament d'un RREP-ACK en resposta a un RREP.

Reservat:
Opcions futures. S'envia com a 0, i és ignorat al destí.

Mida prefix:
Si no és zero, la mida del prefix indica que el *next-hop* indicat pot ser utilitzat per tots els nodes amb el mateix prefix.

Hop Count:
Número de salts que separen el node que reenvia el RREP fins al destí de RREQ.

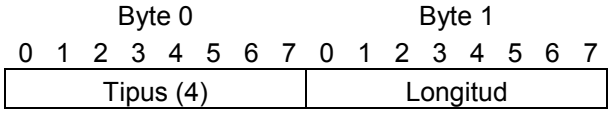
Direcció IP del destí:
Direcció IP del destí pel que es desitja la ruta.

Número de seqüència del destí:
L'últim DSN conegut per al destí per part de l'origen.

Direcció IP de l'origen:
Direcció IP de l'origen que ha sol·licitat la ruta.

Temps de vida:
Temps en ms durant el qual el node que rep el RREP pot considerar la ruta activa

- Route Reply Acknowledgment (RREP-ACK)



Longitud:
Si s'envia com a 0, s'ignora al destí.

- Route Error (RERR)

Byte 0								Byte 1								Byte 2								Byte 3							
0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
Tipus (3)								N	Reservat															Comptador destins							
Direcció IP del destí inassolible 1																															
Número de seqüència del destí inassolible 1																															
Direcció IP del destí inassolible 2																															
Número de seqüència del destí inassolible 2																															

N *No delete flag*; s'activa quan un node ha efectuat un *Local Repair* i no es vol que els nodes precursors esborrin la ruta.

Comptador destins:
Número de destins inassolibles que es descriuen al missatge. Com a mínim 1.

Direcció IP del destí inassolible:
Direcció IP del destí que ja no és assolible.

Número de seqüència del destí inassolible:
Número de seqüència existent a la taula d'encaminament, incrementat en una unitat.

A.2. Expanding Ring Search

Utilitzant aquesta tècnica, el primer RREQ té un TTL amb un valor definit a la variable TTL_START[22]. Es posa en marxa un temporitzador de retransmissió per si no es rep resposta, definit a la variable RING_TRAVERSAL_TIME[23]. Si expira aquest temporitzador i no hem rebut resposta, podem reenviar el RREQ, incrementant-ne el RREQ ID en una unitat. Per a cada nou intent, el TTL augmenta segons el definit a la variable TTL_INCREMENT[22] . Quan el TTL és major que un valor definit a TTL_THRESHOLD[22], el TTL pel pròxim intent s'estableix a NET_DIAMETER. Cada vegada que el node envia un RREQ i no rep resposta, ha d'esperar un temps RING_TRAVERSAL_TIME per tornar a intentar-ho. Aquest temps depèn del TTL actual. El node origen pot generar un màxim de RREQ_RATELIMIT RREQs per segon.

El nombre de reintents addicionals una vegada el TTL ha assolit el valor NET_DIAMETER es defineix a la variable RREQ_RETRIES. Aquest reintents han de seguir un *backoff* exponencial binari (el temporitzador s'inicialitza a NET_TRAVERSAL_TIME[24], i es duplica amb cada retransmissió del RREQ). Si es vol que el primer RREQ arribi al node més llunyà de la xarxa, podem inicialitzar les variables TTL_START i TTL_INCREMENT al valor especificat a NET_DIAMETER.

[22] TTL_START = 1, TTL_INCREMENT = 2, TTL_THRESHOLD = 7 (per defecte)
[23] 2 * NODE_TRAVERSAL_TIME * (TTL_VALUE + TIMEOUT_BUFFER) (per defecte)
[24] 2 * NODE_TRAVERSAL_TIME * NET_DIAMETER (per defecte)

A.3. Gratuitous RREP

Quan s'activa aquesta opció, no canvia el missatge RREP que el node intermedi envia al node origen, però el força a enviar també un altre missatge RREP al destí. El node intermedi simularà la resposta a un RREQ fictici, el que hauria enviat el destí en intentar establir la comunicació bidireccional. Aquest missatge RREP injustificat conté com a *hop count* el valor de l'entrada a la taula de la ruta del node intermedi amb l'origen com a node destí. Al camp d'adreça IP de destí es posa la IP d'origen del RREQ, al camp IP d'origen la IP del node destí del RREQ, el DSN s'estableix al valor OSN del RREQ i el valor de vida de la ruta s'estableix al temps restant de vida de la ruta del node intermedi amb destí cap a l'origen.

Si l'enllaç fos unidireccional, podria ser que el missatge RREP no arribés a l'origen. En aquest cas, al reenviar un RREP es pot fer amb el *flag* A (*acknowledgment*) activat. El node receptor ha de reconèixer aquest paquet. Si no ho fa, el node que havia enviat el RREP l'introdueix a una llista negra. Un node no accepta RREQs dels nodes de la llista negra. Els nodes són esborrats d'aquesta llista expirat un temps BLACKLIST_TIMEOUT.

A.4. Local Repair

El node que detecta el trencament pot intentar reparar la ruta si el node destí no està a més de MAX_REPAIR_TTL salts. Per a fer això, el node intermedi envia un RREQ per al destí de la ruta afectada, incrementant-ne en una unitat el DSN. Si no es rep resposta es un interval de temps PATH_DISCOVERY TIME, s'envia el RERR als nodes precursors tal i com s'ha explicat abans. El node origen segueix enviant dades pensant-se que la ruta segueix activa. Aquestes són guardades realment en el *buffer* del node intermedi, mentre aquest busca una nova ruta. El node que repara la ruta, pot escollir fer el descobriment tant aviat com detecta el trencament, o bé pot esperar a rebre un nou paquet per les rutes afectades. Si ho fa en el primer cas, pot ser que faci un descobriment quan ja no hi ha més dades per enviar.

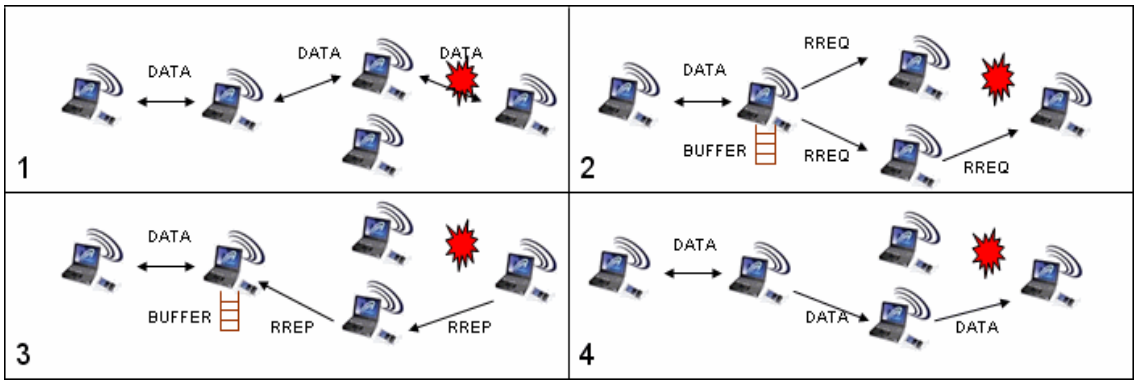


Fig. 2.1 Procés Local Repair.

B. PROBLEMES AMB JADHOC

En un principi, es va pensar de fer servir portàtils i PDAs a les proves realitzades en aquest treball. Hauria estat interessant veure el funcionament del protocol en una xarxa tan heterogènia. Els dos portàtils sempre actuarien com a nodes extrems, i les PDAs com a nodes intermedis, ja que la potència de transmissió dels primers és més elevada. Les PDAs utilitzades eren un model Zaurus SL-5500S, i la implementació de AODV instal·lada en aquest dispositius era JAdhoc.

Es van realitzar unes primeres proves amb aquest escenari, i es van observar molts problemes en el funcionament. Ràpidament va aparèixer la via d'aconseguir quatre portàtils idèntics, equipats ja amb una targeta sense fils. Són els portàtils utilitzats en aquests estudi. Al decidir utilitzar uns altres equips ja no es va analitzar què passava realment amb les Zaurus i JAdhoc.

Els problemes trobats al funcionament de JAdhoc tenien molt a veure amb el protocol ARP. Observem la següent figura.

No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.7.1	255.255.255.255	AODV	Route Request, D: 192.168.7.2, O: 192.168.7.1 Id=0
2	0.319538	192.168.7.1	255.255.255.255	AODV	Route Request, D: 192.168.7.2, O: 192.168.7.1 Id=1
3	0.799465	192.168.7.1	255.255.255.255	AODV	Route Request, D: 192.168.7.2, O: 192.168.7.1 Id=2
4	1.439369	192.168.7.1	255.255.255.255	AODV	Route Request, D: 192.168.7.2, O: 192.168.7.1 Id=3
5	2.240246	192.168.7.1	255.255.255.255	AODV	Route Request, D: 192.168.7.2, O: 192.168.7.1 Id=4
			• • • •		
46	64.451790	192.168.7.1	255.255.255.255	AODV	Route Request, D: 192.168.7.2, O: 192.168.7.1 Id=45
47	65.252667	192.168.7.1	255.255.255.255	AODV	Route Request, D: 192.168.7.2, O: 192.168.7.1 Id=46
48	67.003710	192.168.7.2	Broadcast	ARP	who has 192.168.7.1? Tell 192.168.7.2
49	67.003744	192.168.7.1	192.168.7.2	ARP	192.168.7.1 is at 00:07:eb:31:80:1b
50	67.005875	192.168.7.2	192.168.7.1	AODV	Route Reply, D: 192.168.7.2, O: 192.168.7.1 Hcnt=0
51	67.006328	192.168.7.1	255.255.255.255	AODV	Route Reply, D: 192.168.7.1, O: 192.168.7.1 Hcnt=0
52	67.006504	192.168.7.1	192.168.7.2	ICMP	Echo (ping) request

Fig. B.1 Problemes observats amb JAdhoc.

Com podem veure, el node 192.168.7.2 (PDA), rep missatges AODV del veí 192.168.7.1. Aquest missatges són RREQs generats pel portàtil buscant una ruta per la PDA. Misteriosament, la PDA no respon cap d'aquests paquets, però al cap de aproximadament un minut, sol·licita la direcció MAC del portàtil mitjançant un missatge ARP. El portàtil respon a la sol·licitud amb un altre missatge ARP. Aleshores la PDA respon amb el RREP pertinent i s'inicia la comunicació. No sabem perquè succeeix això. Primer de tot, perquè la PDA no pregunta per la MAC del portàtil fins al cap d'un minut? I per què pregunta per la MAC si a les sol·licituds RREQ rebudes ja apareix aquest valor? Es podria haver fet assignació estàtica de ARP, tal i com s'ha fet en les proves d'aquest treball, però en aquell moment no es coneixia aquesta possibilitat.

El resultat era que quan s'intentaven establir comunicacions al llarg de l'escenari, els veïns no es reconeixien entre ells. Al cap d'una estona tot començava a funcionar, però això era durant poc temps. Per tant, es va deixar de treballar amb aquesta implementació i vam decidir esperar els nous portàtils, per treballar amb una sola implementació de AODV en un escenari homogeni.

C. PROBLEMES AMB ATCP

Aquest ha estat el problema més gran trobat en aquest treball. Molts dels escenaris i proves pensats per a aquest TFC estaven destinats a veure el rendiment real de la implementació de FreeBSD 4.2, fins al punt que aquest treball al principi anava a anomenar-se "Avaluació del nou protocol ATCP (ad-hoc TCP) en una xarxa mòbil ad-hoc real". Finalment, no s'ha pogut instal·lar correctament, sospitant que, de fet, la implementació mai ha arribat a funcionar. A continuació descriurem breument què és el que volíem analitzar i com anava a ser analitzat. Descriurem els passos seguits en la instal·lació del programa, i explicarem com ha evolucionat el treball realitzat.

El primer que podem fer és visitar la pàgina del projecte on podem trobar la única implementació coneguda de ATCP, la web del PACMAN [25]. PACMAN (*Power Aware Computing/Communication for Mobile Ad hoc and Sensor Networks*) ésun projecte realitzat entre les universitats de Califòrnia i Portland, que intenta disminuir el consum energètic dels dispositius sense fils. Aquest projecte en principi està orientat als dispositius que poden cobrir un camp de batalla (no oblidem els origens militars de les xarxes ad-hoc). Per aconseguir-ho, dissenyen algoritmes que redueixen el consum energètic dels dispositius mentre es mantenen les funcionalitats bàsiques d'aquests. El codi font de la implementació per FreeBSD es trobava en aquesta mateixa pàgina en el moment d'iniciar aquest treball. Aquest codi ja no és accessible des d'aquesta web, i que sapiguem, tampoc ho és a qualsevol altre lloc.

El mateix codi font ve acompanyat d'un fitxer README, on es detallen els passos d'instal·lació. S'han seguit els passos del fitxer fil per randa, sobre una instal·lació nova de FreeBSD 4.2, i no hem aconseguit que la implementació funcioni. El primer que veiem al fitxer README és que pot ser que, depenent de l'escenari que vulguem implementar, haurem d'instal·lar abans ALTQ. ALTQ habilita les disciplines de cua a FreeBSD, però en teoria només ha de ser instal·lat si l'equip actua com a *router*. Hem provat de instal·lar ATCP amb ALTQ i sense ALTQ, i no ha funcionat de cap de les dues maneres. Detallem a continuació només els passos seguits per instal·lar ATCP.

ATCP modifica alguns dels arxius que contenen el codi font de TCP de FreeBSD (ubicats a /usr/src/sys/netinet). El que cal és substituir aquest arxius pels existents al directori que conté el codi font del kernel FreeBSD (usr/src/sys). Una vegada fet això, es recompila el kernel amb els nous arxius, i tot hauria de funcionar correctament. Vegem com es fa pas a pas:

Còpia de seguretat dels arxius original de TCP. Creem un directori on guardem els fitxers del codi font original de TCP de FreeBSD.

```
# mkdir /usr/src/sys/netinet/original
# cp /usr/src/sys/netinet/* /usr/src/sys/netinet/original
```

Descomprimim el fitxer atcp-src.tar.gz al directori /atcp-src

```
# tar -zxvf atcp-src.tar.gz /
```

Substituïm els nous arxius de ATCP pels originals de TCP

cp /atcp-src/* /usr/src/sys/netinet

C.1. Recompilar el kernel de FreeBSD

A continuació es descriu en termes genèrics com es recompila el kernel de FreeBSD. Explicarem a cada pas què és el que estem fent, suposant sempre que s'està fent la instal·lació de ATCP sobre una instal·lació nova i neta de FreeBSD.

cd /usr/src/sys/i386/conf
cp GENERIC ATCP_KERNEL

Editem el fitxer ATCP_KERNEL. Modificar la línia ident GENERIC per ident ATCP_KERNEL.

config ATCP_KERNEL

A part dels arxius que contenen el codi font del kernel, existeixen els fitxers de configuració d'aquest. En aquests arxius, s'habiliten les opcions que desitgem que estiguin disponibles en el nostre kernel. És a dir, per a un mateix codi font, podem compilar diferents kernels, amb diferents característiques cadascun. Els fitxers de configuració es troben al directori /usr/src/sys/conf/i386. En el nostre cas, podem copiar el arxiu de configuració GENERIC de FreeBSD per compilar ATCP, ja que les opcions requerides són les mateixes, i no en necessitem cap addicional. L'únic que hem de fer és canviar el identificador per al nou kernel, tal i com s'ha explicat abans.

Una vegada hem creat el nou fitxer de configuració per al kernel ATCP, executem *config* per preparar els fitxers que serviran per compilar el nou kernel. Al cap de poca estona, se'ns dirà que ja podem anar al directori /usr/src/sys/compile/ATCP_KERNEL. Es tracta de que el nou codi font sigui compilat per poder funcionar com un nou kernel. Per a fer això, cal executar:

Crear i comprovar les dependències del kernel, i posteriorment e*sborrar fitxers generats no necessaris.*

make depend
make clean

Compilar i linkar el kernel i els mòduls. Posteriorment, l'instal·lem com a kernel per defecte i reiniciem.

make all
make install
sync
shutdown -r now

C.2. Modificacions realitzades en el codi de ATCP

Tal i com ja hem comentat, ATCP finalment no ha funcionat. Però ja és estrany que, fent la instal·lació pas a pas, tal i com s'indica en el fitxer README, sobre una instal·lació fresca de FreeBSD 4.2, el programa no compilés correctament. Si seguim els passos mostrats anteriorment, el primer error l'obtindrem a l'hora de linkar el kernel. Se'ns mostra el següent missatge:

in_proto.o (.data+0x74): undefined reference to 'atcp_input'

Aquest error té fàcil solució. La funció atcp_input està definida en el fitxer atcp.h. Si afegim aquest als *includes* del fitxer in_proto.c, la compilació del nou kernel es durà a terme sense cap problema. De totes maneres, encara que el kernel es compili correctament, no es pot establir cap connexió TCP. En el moment d'establir el *three way handshake* de TCP, obtenim un error intern del kernel de Linux. Aquests problemes són gairebé impossibles de depurar, així que no hem pogut fer res al respecte. L'error obtingut és el següent:

```
Fatal trap 12: page fault while in kernel mode
fault virtual address = 0x0
fault code = supervisor read, page not present
instruction pointer = 0x8:0x0
stack pointer = 0x10:0xc0372870
frame pointer = 0x10:0xc0372890
code segment = base 0x0, limit 0xfffff, type 0x1b
                    = DPL 0, pres 1 def32 1 ,gran 1
processor eflags = interrupt enabled, rsume, IOPL = 0
current process = idle
interrupt mask =
trap number =12
panic: page fault

rebooting....
```

Veient aquest error, s'ha provat de instal·lar també ALTQ. La instal·lació d'aquesta aplicació es bastant més complicada que la de ATCP, i l'error obtingut al final era el mateix. Només remarcarem que s'obtenen nous errors en el codi de ATCP, ja que en aquest cas s'avalua tot el codi de la implementació de ATCP, no com abans. La modificació que hem de fer a ATCP si decidim provar d'instal·lar també ALTQ és afegir les següents línies al fitxer atcp.h (incloent aquest en tcp_input.c i tcp_output.c):

#define TH_ECNECHO 0x40
#define TH_CWR 0x80

Hem estat en contacte amb un dels coautors de la implementació (Harkirat Singh), i després de mantenir un intens diàleg, no va saber explicar què era el que passava. Sospitant ja que la implementació mai ha funcionat, es va enviar un missatge a MANET intentant esbrinar si algú l'havia fet servir alguna vegada. A dia d'avui, encara no s'ha rebut cap resposta, el que fa pensar que no anàvem gaire desencaminats en les nostres suposicions.

C.3. Com escollir amb quin kernel treballar

El kernel el podem escollir en el moment d'iniciar el sistema, no una vegada aquest hagi arrencat. Amb FreeBSD s'instal·la un gestor d'inici del sistema, des del que podrem escollir amb quin kernel volem treballar. Aquest gestor, però, no és tan gràfic i fàcil d'utilitzar com poden ser les versions més modernes de *grub* i *lilo*, els gestors més comuns en els sistemes Linux, ja que no hi apareix cap menú.

En el moment de compilar un nou kernel, si executem *make install,* aquest serà el kernel per defecte en el pròxim inici del sistema. El fitxer que conté aquest kernel és /kernel, i si aquest existia prèviament, és renombrat a /kernel.old. És una bona idea que sempre que compilem un nou kernel, renombrem el kernel actual per un nom que l'identifiqui (p.e: kernel.GENERIC, kernel.ATCP). Així, no serà renombrat a /kernel.old, ja que podria ser que existís també i seria esborrat definitivament.

És molt possible que, com en el nostre cas, en algun moment haguem de tornar a fer servir un kernel anterior. Per fer-ho, tenim dues opcions, editar un fitxer on modificar el kernel per defecte o escollir-lo manualment en el moment d'iniciar el sistema. Si volem modificar el kernel per defecte, haurem d'editar el fitxer /boot/defaults/loader.conf. Aquest fitxer té per defecte establerts permisos de només lectura. Podem canviar aquests permisos, modificar al fitxer, i restablir els permisos originals

chmod 777 /boot/defaults/loader.conf

Editar la línia del fitxer loader.conf kernel="kernel" per kernel="nom_i_ruta_nou_kernel_per_defecte" .

chmod 444 /boot/defaults/loader.conf

En el proper inici, "*nom_i_rutanou_kernel_per_defecte" serà* el kernel utilitzat. En el moment que s'inicia el sistema operatiu, es mostra un comptador. Si transcorregut el temps indicat pel comptador no hem polsat cap tecla, s'iniciarà el kernel per defecte, però si hem polsat qualsevol tecla que no sigui Intro, accedirem al menú del gestor d'inici. Per iniciar el kernel que nosaltres vulguem en aquest moment, cal executar:

unload
boot *nom_i_ruta_del_kernel*

La primera comanda és requerida pel propi sistema per descarregar els mòduls del kernel per defecte. Una vegada fet això, ja podem iniciar amb el nou kernel.

D. EL KERNEL DE LINUX

El kernel és el centre d'operacions en un sistema Linux. La veritat és qui si s'aconsegueix configurar un kernel, i aquest funciona correctament, no ha de ser sempre necessari actualitzar a la última versió. Existeixen les versions estables i les versions en desenvolupament. Les primeres tenen un funcionament realment garantit, mentre que les segones poden donar suport a noves característiques però tenir errarades en altres parts del codi. En el nostre cas, hem optat per instal·lar la última versió estable del kernel de Linux en el moment de realitzar el treball, el kernel 2.6.9.

El primer que hem de fer és aconseguir el codi font del kernel de Linux [26]. Aquest pot ser descarregat gratuïtament des de Internet. Una vegada tenim el codi font (en format tar.gz), el podem descomprimir al directori /usr/src/linux-2.6.9. La configuració del kernel es pot fer de tres maneres, sempre executant les següents comandes des del directori on es troba el codi font del kernel a compilar:

- make config: configurar per línia de comandes
- make menuconfig: configurar amb un menú en mode text
-make xconfig: configurar des d'un menú gràfic (requereix el sistema X operatiu)

Només comentarem el segon mètode, ja que és el sistema utilitzat normalment. No entrarem en detalls, perquè el menú ja disposa d'un sistema d'ajuda el prou complert. A la següent figura podem veure una captura d'aquest menú.

Fig. D.1 Menú de configuració del kernel.

Com es pot veure, totes les opcions del kernel poden ser modificades des d'aquest menú. Aquestes opcions estan organitzades en diferents seccions. Recomanem navegar per aquestes i consultar l'ajuda del menú. La configuració per defecte sol ser l'adequada per a un sistema estàndard (normalment situada al fitxer ocult /usr/src/linux-2.x.x/.config).

Bàsicament, el menú de configuració del kernel ens serveix per habilitar els mòduls que nosaltres vulguem. Un mòdul pot ser un *driver* per un dispositiu (com ho és IPW2100) o qualsevol altre programa (p.e: el mòdul de AODV-UU), però només pot ser cridat des del propi kernel (no és un executable normal i corrent). Els mòduls poden ser carregats directament en el moment d'iniciar el sistema, o bé carregats només quan el kernel ho creu oportú. Si es carreguen al iniciar el sistema, pot ser que estiguem carregant mòduls que no siguin necessaris, malgastant recursos. També poden ser instal·lats posteriorment a la compilació d'un kernel. El mode de funcionament es defineix amb un flags. Ho podem veure a la següent taula:

Taula D.1 Selecció del comportament dels mòduls

Mòdul carregat amb l'inici del sistema	[*]
Mòdul cridat pel kernel només quan és requerit	<M>
Mòdul no disponible	[ ]

La veritat és que hi ha moltes opcions disponibles en aquest menú. Com ja hem esmentat, potser la configuració per defecte funciona en el nostre sistema, però si mirem les opcions una per una, aconseguirem que el nostre kernel funcioni millor i que no es malgastin els recursos del sistema.

Una vegada la configuració es la correcta, podem seguir els següents passos per compilar el nou kernel. Les comandes que escrivim a continuació només són vàlides per a la nova branca del kernel, la 2.6.x:

```
# make all
# make install
# make clean
```

Per compilar un kernel 2.4.x, els passos a seguir després de la configuració són:

```
# make dep
# make modules && make modules_install
# make && make bzImage
# make install
```

D'aquesta manera, es compilarà el nou kernel. En aquest cas, el nou kernel serà instal·lat i configurat correctament en el gestor d'inici (el gestor d'inici de Fedora és *grub*), sense cap acció requerida addicional. Una vegada reiniciem el sistema, se'ns mostrarà un menú en el que podrem escollir quin kernel volem executar.

E. SCRIPTS DE CONFIGURACIÓ

E.1. Escenaris en línia

PC192.168.7.x

- Configurar xarxa ad-hoc (comandes incloses a l'arxiu /root/inici):

```
# modprobe ipw2100
# ifconfig eth0 192.168.7.2 netmask 255.255.255.0
# iwconfig eth0 mode Ad-Hoc
# iwconfig eth0 essid ATCP
# iwconfig eth0 channel 1
# arp -f
# ifconfig
# iwconfig
```

- Configurar iptables per emuular àrees de cobertura desitjades (comandes incloses a l'arxiu /root/iptablesping)

```
PC 192.168.7.2 :
# iptables -t filter -A INPUT -m mac --mac-source 00:0C:F1:10:6B:A4 -j DROP
# iptables -t filter -A INPUT -m mac --mac-source 00:0C:F1:2A:38:EE -j DROP
# iptables -t filter -A INPUT -m mac --mac-source 00:07:EB:31:80:1B  -j DROP

PC 192.168.7.3 :
# iptables -t filter -A INPUT -m mac --mac-source 00:0C:F1:2A:38:EE -j DROP
# iptables -t filter -A INPUT -m mac --mac-source 00:07:EB:31:80:1B  -j DROP

PC 192.168.7.4 :
# iptables -t filter -A INPUT -m mac --mac-source 00:0C:F1:2A:3B:29 -j DROP
# iptables -t filter -A INPUT -m mac --mac-source 00:07:EB:31:80:1B  -j DROP

PC 192.168.7.5 :
# iptables -t filter -A INPUT -m mac --mac-source 00:0C:F1:2A:3B:29 -j DROP
# iptables -t filter -A INPUT -m mac --mac-source 00:0C:F1:0D:EB:6A  -j DROP

PC 192.168.7.6 :
# iptables -t filter -A INPUT -m mac --mac-source 00:0C:F1:2A:3B:29 -j DROP
# iptables -t filter -A INPUT -m mac --mac-source 00:0C:F1:0D:EB:6A  -j DROP
# iptables -t filter -A INPUT -m mac --mac-source 00:0C:F1:10:6B:A4 -j DROP
```

E.2. Escenaris amb més d'un camí al destí

PC192.168.7.x

- Configurar xarxa ad-hoc (comandes incloses a l'arxiu /root/inici):

```
# modprobe ipw2100
# ifconfig eth0 192.168.7.2 netmask 255.255.255.0
# iwconfig eth0 mode Ad-Hoc
# iwconfig eth0 essid ATCP
# iwconfig eth0 channel 1
# arp -f
# ifconfig
# iwconfig
```

A més, en el portàtil que actua com a gateway (192.168.7.2), cal afegir-hi la configuració per la xarxa Ethernet.

```
# modprobe b44
# ifconfig eth1 192.168.6.2 netmask 255.255.255.0
```

- Configurar iptables per emuular àrees de cobertura desitjades (comandes incloses a l'arxiu /root/iptablesgap)

PC 192.168.7.2 :
```
# iptables -t filter -A INPUT -m mac --mac-source 00:0C:F1:2A:38:EE -j DROP
```

PC 192.168.7.3 :
```
# iptables -t filter -A INPUT -m mac --mac-source 00:0C:F1:10:6B:A4 -j DROP
```

PC 192.168.7.4 :
```
# iptables -t filter -A INPUT -m mac --mac-source 00:0C:F1:0D:EB:6A  -j DROP
```

PC 192.168.7.5 :
```
# iptables -t filter -A INPUT -m mac --mac-source 00:0C:F1:2A:3B:29 -j DROP
```

F. IPTABLES

Iptables és l'eina amb la que podem configurar les regles de filtrat d'un dispositiu, creant un tallafocs o *firewall*. Podem acceptar o rebutjat informació segons la MAC o la IP d'origen o fins i tot del port utilitzat a nivell de transport. Una altra de les seves característiques és poder configurar un *Network Address Translation* (NAT).

Per disposar de *iptables*, necessitem tenir instal·lat l'entorn de filtrat de paquets *Netfilter*.

Els conceptes *bàsics* de iptables són:

- Taules: ofereixen certes funcionalitats. Les taules per defecte són *filter*, *nat* i *mangle*.
- Cadenes: una cadena és el camí que pot seguir un paquet. Taules diferents contenen cadenes diferents. Quan un paquet travessa una cadena, pot ser acceptat o rebutjat segons la política per defecte establert a la cadena.
- Regles: la política a seguir per a cada cadena.

La taula *filter* és la utilitzada per defecte, i les seves cadenes per defecte són INPUT, FORWARD i OUTPUT. INPUT només s'aplica a la informació destinada a l'equip local, FORWARD a la informació amb origen i destí diferent de la de l'equip local, i OUTPUT per a la informació generada a l'equip local.

La taula *nat* és consultada quan es troba un datagrama que crea una nova connexió, i per descomptat quan es vol configurar un NAT. Les cadenes per defecte son PREROUTING, POSTROUTING i OUTPUT. La cadena PREROUTING altera la informació tant bon punt entren a l'equip local, la cadena POSTROUTING l'altera just en el moment de ser reenviada, i OUTPUT modifica la informació generada localment just abans de ser encaminada.

La taula *mangle* s'encarrega de l'alteració especialitzada de la informació. Les cadenes per defecte són PREROUTING i OUTPUT.

La sintaxis general de les comandes de *iptables* és la següent:

iptables -t *taula opció especificació-regla1 especificació-regla2*...

Les opcions disponibles són:

-P *nom_cadena política* (canviar la política a seguir, que pot ser DROP per rebutjar o ACCEPT per acceptar)
-L *nom_cadena* (veure les regles de la cadena)
-F *nom_cadena* (esborrar les regles de la cadena)
-A nom_cadena (afegir una cadena de la taula)
-R *nom_cadena* (substituir una cadena de la taula)
-I *nom_cadena* (inserir una cadena de la taula)

Les especificacions de les regles poden ser moltes, i es poden utilitzar més d'una a la vegada. En llistem les més interessants:
-p (protocol: TCP, UDP, ICMP, etc.)
-s (direcció origen)
-d (direcció destí)
-i (interfície d'entrada)
-o (interfície de sortida)
--dport (port destí)
--sport (port origen)
--mac-source (direcció MAC origen)
–j (especifica l'acció a seguir quan la informació coincideix amb la regla)
ACCEPT (acceptar el paquet)
DROP (eliminar el paquet)
REJECT (eliminar el paquet, informant al emissor)
LOG (fer un log del paquet, i res més)

Altres accions:

iptables -v (mostra les sortides en mode text)
iptables -V (mostra les versió instal·lada)
iptables -h (mostra les opcions generals del programa)

G. ROUTE

La comanda *route* serveix bàsicament per llistar o manipular la taula d'encaminament del sistema. Això en és de gran utilitat a l'hora de definir les rutes estàtiques dels nostres escenaris.

La sintaxi general de l'aplicació és:

route *opció paràmetre1 paràmetre2...*

Les opcions bàsiques són *add*, que ens servirà per afegir una nova entrada a la taula d'encaminament, i *del*, que ens servirà per esborrar una entrada ja existent. Si no s'utilitza cap de les dues opcions, només es llistarà el contingut de la taula per pantalla.

Els paràmetres que accepta la comanda *route* són els següents:

-net (per agregar o eliminar una xarxa destí)
-host (per agregar un equip individual com a destí)
netmask (per especificar un prefix de xarxa)
gw (per especificar la porta d'enllaç ca a una xarxa)
dev (per assignar la ruta a una interfície de xarxa determinada)

Altres opcions disponibles són:

route-v (mostra les sortides en mode text).
route -V (mostra les versió instal·lada de *net-tools*, aplicació que conté route).
route -h (mostra les opcions generals del programa)
route -F (operar sobre la taula d'encaminament del kernel)
route -C (operar sobre la memòria cau d'encaminament del kernel)
route -h (mostra les opcions generals del programa)

A continuació mostrem alguns exemple:

route add -net 192.168.7.0 netmask 255.255.255.0 dev eth0
route add -net 192.168.7.0/24 dev eth0

Aquestes dues comandes són iguals. Els paquets dirigits a la xarxa 192.168.7.0/24 sortiran per la interfñicie de xarxa eth0.

route add -net 192.168.7.0 netmask 255.255.255.0 gw 192.168.6.1

Amb aquesta comanda, els paquets dirigits a la xarxa 192.168.7.0 són enviats al equip 192.168.6.1A FreeBSD, la sintaxi varia lleugerament. Per realitzar l'acció anterior, la sintaxi és la següent:

route add 192.168.7.0 192.168.6.1 255.255.255.0

H. AODV-UU

H.1. Codi que modifica el forwarding del node i deshabilita els ICMP

En el codi de main.c trobem la següent funció:

```
int set_kernel_options()
{
    int i, fd = -1;
    char on = '1';
    char off = '0';
    char command[64];

    if ((fd = open("/proc/sys/net/ipv4/ip_forward", O_WRONLY)) < 0)
      return -1;
    if (write(fd, &on, sizeof(char)) < 0)
      return -1;
    close(fd);

...

    /* Disable ICMP redirects on all interfaces: */

memset(command, '\0', 64);
       sprintf(command, "/proc/sys/net/ipv4/conf/%s/send_redirects",
              DEV_NR(i).ifname);
       if ((fd = open(command, O_WRONLY)) < 0)
           return -1;
       if (write(fd, &off, sizeof(char)) < 0)
           return -1;
       close(fd);
       memset(command, '\0', 64);
       sprintf(command, "/proc/sys/net/ipv4/conf/%s/accept_redirects",
              DEV_NR(i).ifname);
       if ((fd = open(command, O_WRONLY)) < 0)
           return -1;
       if (write(fd, &off, sizeof(char)) < 0)
           return -1;
       close(fd);
    }
    memset(command, '\0', 64);
    sprintf(command, "/proc/sys/net/ipv4/conf/all/send_redirects");
    if ((fd = open(command, O_WRONLY)) < 0)
      return -1;
    if (write(fd, &off, sizeof(char)) < 0)
      return -1;
    close(fd);


    memset(command, '\0', 64);
    sprintf(command, "/proc/sys/net/ipv4/conf/all/accept_redirects");
    if ((fd = open(command, O_WRONLY)) < 0)
      return -1;
    if (write(fd, &off, sizeof(char)) < 0)
      return -1;
    close(fd);

    return 0;
}
```

H.2. El protocol ARP

Amb aquest protocol, els nodes poden preguntar amb un missatge *broadcast* a tots els nodes de la xarxa quina MAC és la propietària d'una IP determinada, si no coneixen aquesta relació. Tots els nodes que reben aquesta petició sabran aleshores la relació MAC-IP[25] per a l'origen, ja que poden deduir aquesta informació del missatge de petició, i per tant, podran dirigir-se a ell. Només el node propietari de la IP sol·licitada al primer missatge ARP respon a l'origen, amb un missatge ARP que conté la seva direcció MAC. El node origen disposa aleshores de la informació necessària per enviar informació al destí.

Aquest fenomen afectarà al temps d'anada i tornada del primer paquet, ja que abans d'enviar la primera sol·licitud de ping (*ping_reply*), el node origen haurà d'esbrinar quina MAC té el node del que espera resposta. Fins que no es coneix aquesta informació, el node origen no pot enviar informació al destí. El temps que es triga en conèixer aquesta informació és contabilitzat en el primer dels paquets de la sèrie de pings. S'obté aleshores per al primer dels retards un valor aproximadament el doble que amb les següents sol·licituds.

Aquest problema es pot solucionar assignant estàticament el valor d'aquestes parelles. En el nostre cas, els nodes no canvien la seva direcció IP, així que coneixem en tot moment la MAC que tenen assignada. Per a fer aquesta assignació de manera manual i estàtica, podem utilitzar l'eina *arp*, inclosa amb la distribució Fedora Core 2. Aquesta eina permet llegir el valor d'aquestes parelles d'un fitxer. Per defecte, les parelles s'escriuen en el fitxer /etc/ethers, encara que en podem especificar un altre si ho desitgem. Per llegir aquest fitxer i evitar que els nodes hagin de preguntar als seus veïns per la seva MAC, hem d'executar des del terminal de tots els nodes la comanda arp –f.

El contingut d'aquest fitxer per a cadascun dels portàtils el podem veure a la següent taula:

Taula H.1 Contingut del fitxer /etc/ethers dels diferent nodes

192.168.7.2	192.168.7.3	192.168.7.4
192.168.7.3 00:0C:F1:0D:EB:6A 192.168.7.4 00:0C:F1:10:6B:A4 192.168.7.5 00:0C:F1:2A:38:EE 192.168.7.6 00:07:EB:31:80:1B	192.168.7.2 00:0C:F1:2A:3B:29 192.168.7.4 00:0C:F1:10:6B:A4 192.168.7.5 00:0C:F1:2A:38:EE 192.168.7.6 00:07:EB:31:80:1B	192.168.7.2 00:0C:F1:2A:3B:29 192.168.7.3 00:0C:F1:0D:EB:6A 192.168.7.5 00:0C:F1:2A:38:EE 192.168.7.6 00:07:EB:31:80:1B

192.168.7.5	192.168.7.6
192.168.7.3 00:0C:F1:0D:EB:6A 192.168.7.4 00:0C:F1:10:6B:A4 192.168.7.5 00:0C:F1:2A:38:EE 192.168.7.6 00:07:EB:31:80:1B	192.168.7.2 00:0C:F1:2A:3B:29 192.168.7.3 00:0C:F1:0D:EB:6A 192.168.7.4 00:0C:F1:10:6B:A4 192.168.7.5 00:0C:F1:2A:38:EE

[25] Aquesta informació es guarda durant un cert període de temps a la memòria cau dels ordinadors.

H.3. Temps de procés dels paquets AODV

Ja hem explicat en el treball, en el apartat del RTT obtingut amb AODV, que el el RTT obtingut pel primer paquet, el que inclou el descobriment, idealment hauria de ser el doble que el RTT obtingut pels següents paquets. Això no és cert, ja que els nodes triguen un cert temps en processar els paquets. AODV-UU treballa en l'espai d'usuari. Això provoca que el temps de procés als nodes sigui major, en teoria, que si s'hagués utilitzat una implementació que modifiqués el propi kernel. A continuació farem una estimació d'aquest temps de procés.

1 SALT: En un principi, hem vist quins són els temps de procés relatius a AODV en el escenari en línia de 2 salts. A la gràfica es port veure exactament quins són els temps mesurats, amb l'ajuda d'*Ethereal*.

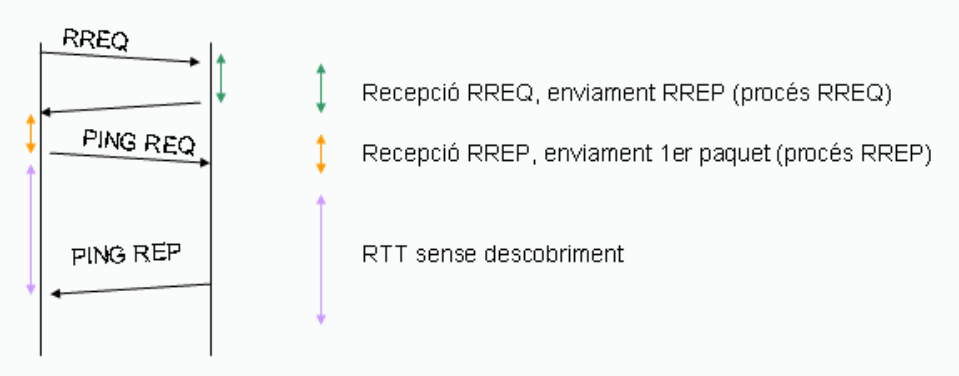


Fig. H.1 Temps de procés de AODV en el escenari de 1 salt

Aquests han estat els resultats obtinguts:

Taula H.2 Temps procés dels paquets AODV en escenaris de 1 salt

Test	Procés RREQ (µs)	Procés RREP (µs)
1	815	662
2	805	687
3	785	4202
4	808	1908
5	765	651
6	782	694
7	793	667
8	784	654
9	793	661
10	783	685
Mitja	791	1105

Tenint en compte que, de les proves de l'apartat 6.2.1, per 1 salt:

RTT amb descobriment (ms)	6,69
RTT sense descobriment (ms)	1,94

Aproximem:

RTT amb descobriment = 2 * RTT sense descobriment+ procés = 2 * 1,94 + 0,791 + 1,15 = 5,82 ms.

El valor s'aproxima bastant al que hem obtingut a la pràctica per al RTT amb descobriment.

2 SALTS: A continuació, s'ha realitzat el mateix estudi que abans però amb l'escenari de 3 nodes. Apareixen nous temps que no existien en el escenari anterior.

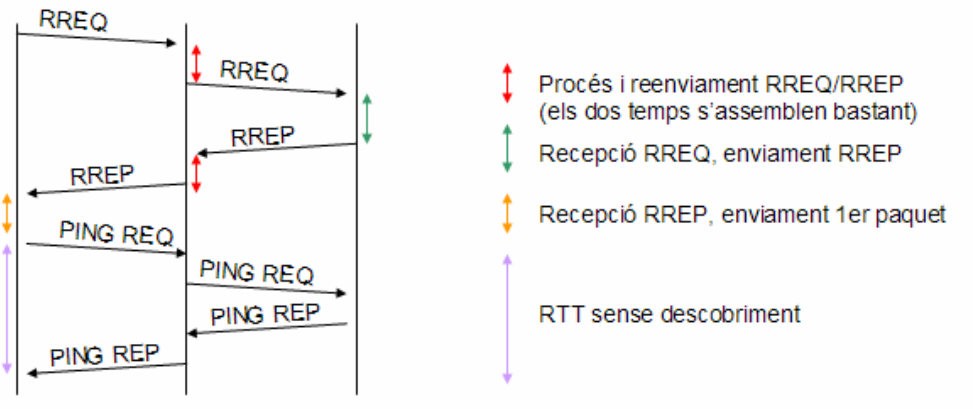


Fig H.2 Temps de procés de AODV en el escenari de 2 salts

Aquests han estat els resultats obtinguts:

Taula H.3 Temps procés dels paquets AODV en escenaris de 2 salts

Test	reenviament RREQ (µs)	procés RREQ (µs)	reenviament RREP (µs)	procés RREP (µs)
1	735	1062	580	619
2	668	1079	582	623
3	659	1072	576	631
4	682	1067	648	635
5	669	1073	595	619
6	695	1385	599	619
7	678	1074	584	623
8	704	1084	776	624
9	693	1673	590	621
10	657	1072	597	619
Mitja	684	1164	612	623

Tenint en compte que, de les proves de l'apartat 6.2.1, per 2 salts:

RTT amb descobriment (ms)	11,46
RTT sense descobriment (ms)	4,05

Aproximem:

RTT amb descobriment = 2 * RTT sense descobriment + procés = 2 * 4,05 + 0,684 + 1,164 + 0,612 + 0,623= 11,183 ms.

Una altre cop, el valor s'aproxima bastant al que hem obtingut a la pràctica.

3 SALTS: A continuació, s'ha realitzat el mateix estudi que abans però amb l'escenari de 4 nodes. Apareixen nous temps que no existien en el escenari anterior.

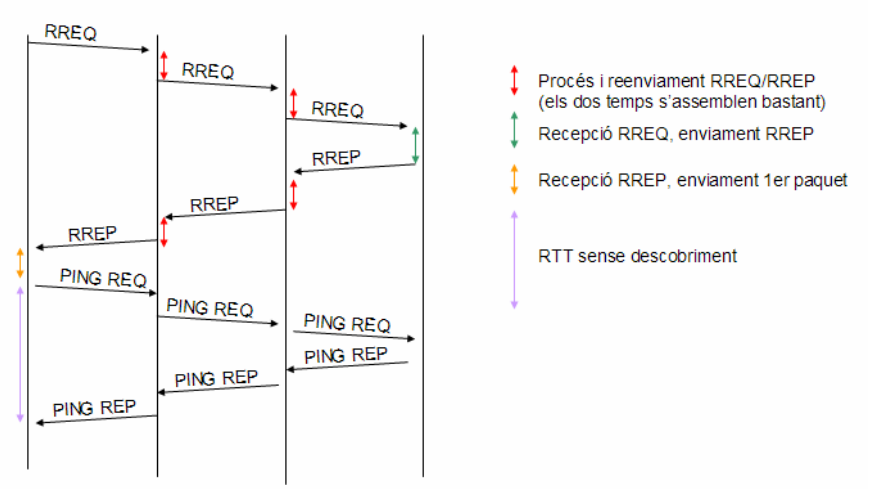


Fig. H.3 Temps de procés de AODV en el escenari de 3 salts

Aquests han estat els resultats obtinguts:

Taula H.4 Temps procés dels paquets AODV en escenaris de 3 salts

Test	reenviament RREQ node 1 (µs)	reenviament RREQ node 2 (µs)	procés RREQ (µs)	reenviament RREP node 1 (µs)	reenviament RREP node 2 (µs)	procés RREP (µs)
1	701	995	1071	574	552	4004
2	704	982	1069	583	548	5509
3	708	988	1072	476	561	2205
4	664	981	1076	565	517	646
5	703	985	1076	619	555	659
6	690	998	1078	563	547	648
Mitja	695	988	1073	563	546	2278

Tenint en compte que, de les proves de l'apartat 6.2.1, per 3 salts:

RTT amb descobriment (ms)	22,23
RTT sense descobriment (ms)	5,94

Aproximem:

RTT amb descobriment = 2 * RTT sense descobriment + procés = 2 * 5,94 + 0,695 + 0,998 + 1,073 + 0,563 + 0,546 + 2,278 = 18,033 ms.

El valor s'aproxima bastant, però no tant com abans, al que hem obtingut a la pràctica. En el cas de 3 salts apareixen temps de procés injustificables, sobretot en la recepció del RREP fins que es transmet el primer paquet. D'aquí ve la variació del resultat.

4 SALTS: A continuació, s'ha realitzat el mateix estudi que abans però amb l'escenari de 5 nodes. Apareixen nous temps un altre cop.

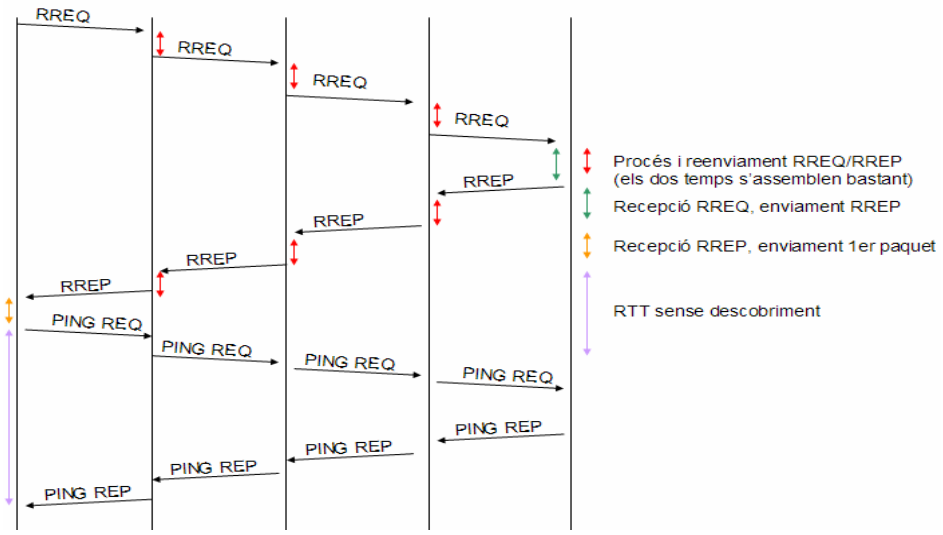


Fig. H.4 Temps de procés de AODV en el escenari de 5 salts

Aquests han estat els resultats obtinguts:

Taula H.5 Temps procés dels paquets AODV en escenaris de 5 salts

Test	reenviament RREQ node 1 (µs)	reenviament RREQ node 2 (µs)	reenviament RREQ node 3 (µs)	procés RREQ (µs)	reenviament RREP node 1 (µs)	reenviament RREP node 2 (µs)	reenviament RREP node 3 (µs)	procés RREP (µs)
1	699	982	997	1080	597	531	579	654
2	690	987	976	1086	565	541	543	1216
3	688	985	987	1072	550	528	533	3582
Mitja	692	984	986	1079	570	533	551	1817

Tenint en compte que, de les proves de l'apartat 6.2.1, per 4 salts:

RTT amb descobriment (ms)	26,01
RTT sense descobriment (ms)	7,95

Aproximem:

RTT amb descobriment = 2 * RTT sense descobriment + procés = 2 * 7,95 + 0,692 + 0,984 + 0,986 + 1,079 + 0,57 + 0,533 + 0,551 + 1,817 = 23,117 ms.

El valor s'aproxima bastant, però no tant com abans, al que hem obtingut a la pràctica per al RTT amb descobriment. En el cas de 4 salts apareixen els mateixos temps estranys de procés.

Dels resultats obtinguts en aquestes proves, hem trobat un error força important en el funcionament de AODV-UU. Curiosament, en el descobriment de ruta, una vegada el RREQ ja arribat al destí, i el RREP viatja de camí, aquest últim desapareix moltes vegades en el precursor del destí. Aquest node rep el RREP del destí, però no el retransmet al seu node precursor. Això provoca que expiri el temporitzador de retransmissió de RREQ a l'origen, que es veu forçat a fer un altre descobriment. Aquest nou descobriment no arriba al destí, ja que el precursor d'aquest ja en coneix la ruta, i en aquesta segona petició sí que respon al seu precursor. El fet, però, que el node origen hagi de fer un nou descobriment, fa que el descobriment en aquest cas sigui molt més gran de l'esperat (recordem com es calculen aquestes retransmissions). Ho podem veure a la següent figura.

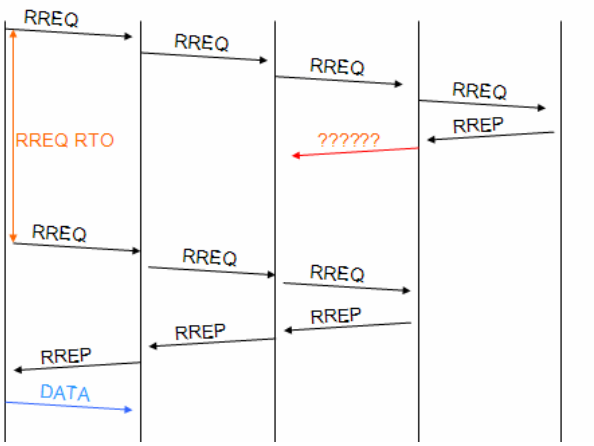


Fig. H.5 Mal comportament del node precursor al destí

Hem informat a la llista de correu de MANET sobre aquest mal funcionament, a veure si algú sabia què estava passant. La gent no sabia que era el que passava, i fins i tot ens va respondre l'Erik Nordström, el creador de AODV-UU. No se sap l'origen d'aquest problema. Degut al temps que triga el node origen en retransmetre el RREQ, obtenim temps molt més grans per als primers pings, els que inclouen descobriment. Els resultats obtinguts pels pings amb descobriment en aquest cas es mostren a la següent taula. Només veiem variacions en els casos de 2, 3 i 4 salts, ja que en el cas de 1 salt el node precursor és el mateix origen.

Taula H.6 RTT amb descobriment, amb error en el funcionament.

	1 salt	2 salts	3 salts	4 salts
Mitja RTT (ms)	6,69	446,93	573,87	659,07

H.4. Intervals de *hello* reals observats

Per a valors molt petits de HELLO_INTERVAL, concretament inferiors a 200 ms, el valor observat a la realitat és major que el configurat al fitxer params.h. A continuació mostrem i analitzem aquests intervals de *hello*.

No.	Time	Source	Destination	Protocol
1	0.000000	192.168.7.2	255.255.255.255	AODV
4	0.000081	192.168.7.2	255.255.255.255	AODV
7	0.000162	192.168.7.2	255.255.255.255	AODV
11	0.000252	192.168.7.2	255.255.255.255	AODV
16	0.000363	192.168.7.2	255.255.255.255	AODV
20	0.000452	192.168.7.2	255.255.255.255	AODV
24	0.000541	192.168.7.2	255.255.255.255	AODV
26	0.000586	192.168.7.2	255.255.255.255	AODV
32	0.000721	192.168.7.2	255.255.255.255	AODV
34	0.000771	192.168.7.2	255.255.255.255	AODV
38	0.000871	192.168.7.2	255.255.255.255	AODV
43	0.000983	192.168.7.2	255.255.255.255	AODV

Fig. H.6 Interval de *hello* reals per HELLO_INTERVAL=50

Si configurem HELLO_INTERVAL a 50ms, veiem com els missatges de *hello* es generen realment, en mitja, cada 80-85 ms. Mentre la diferència entre dos *hello*s consecutius no sigui major a ALLOWED_HELLO_LOSS * HELLO_INTERVAL, no tindrem problemes amb la connectivitat amb els veïns. En aquest cas, podem veure com entre els *hello*s dels paquets 11 i 16 transcorren 111 ms. Amb la configuració del RFC, en aquest exemple, els veïns detectaran un trencament de l'enllaç, ja que 111 ms > 100 ms.

A continuació veiem què succeeix configurant HELLO_INTERVAL a 100 ms.

No.	Time	Source	Destination	Protocol
4	0.000085	192.168.7.2	255.255.255.255	AODV
8	0.000175	192.168.7.2	255.255.255.255	AODV
12	0.110682	192.168.7.2	255.255.255.255	AODV
16	0.218573	192.168.7.2	255.255.255.255	AODV
20	0.333553	192.168.7.2	255.255.255.255	AODV
24	0.472559	192.168.7.2	255.255.255.255	AODV
28	0.618213	192.168.7.2	255.255.255.255	AODV
32	0.750494	192.168.7.2	255.255.255.255	AODV
36	0.869458	192.168.7.2	255.255.255.255	AODV
40	1.010435	192.168.7.2	255.255.255.255	AODV
44	1.140425	192.168.7.2	255.255.255.255	AODV
49	1.285403	192.168.7.2	255.255.255.255	AODV

Fig. H.7 Intervals de *hello* reals per HELLO_INTERVAL=100

En aquest cas, amb la configuració del RFC, tindrem problemes si entre os *hello*s consecutius transcorren més de 200 ms. De la gràfica, podem extraure que en aquest cas la mitja real en el interval de *hello* és aproximadament al voltant de 150 ms, però en cap cas transcorren 200 ms. Del estudi realitzat en el treball, però, deduïm que aquest problemes sí han existit.

En les traces de les figures anteriors, els nodes no transmetien cap tipus de informació que no fossin aquests missatges de *hello* (recordem el funcionament per defecte de AODV-UU, els *hello*s s'envien per defecte encara que no existeixin rutes actives). La raó és que aquests temps es veuen incrementats quan transmetem un flux UDP saturant el medi, com en el cas de les proves d'ample de banda. En el cas de TCP, això no succeeix, ja que la transmissió és controlada i existeixen la finestra i el control de congestió. Cursant UDP, els nodes no detecten els veïns en certs instants de temps, el que provoca trencaments en els enllaços. A causa d'aquests trencaments, es perden molts paquets que no arriben al destí. Aleshores, per a valors baixos de interval de *hello*, *iperf* ens indica que el ample de banda és menor que el que hauríem d'esperar. Deduïm que això succeeix degut a la prioritat de sortida dels paquets a la cua dels nodes. Quan el node té tanta informació a transmetre, els intervals de *hello* es veuen encara més distorsionats, potser perquè el node intenta transmetre el gran cabdal UDP i no té en compte la importància dels missatges de control AODV (no els prioritza). Veiem una altra traça amb HELLO_INTERVAL=100s, però en aquest cas, el node cursa el tràfic UDP saturant el medi.

No.	Time	Source	Destination	Protocol
1	0.000000	192.168.7.2	255.255.255.255	AODV
27	0.237690	192.168.7.2	255.255.255.255	AODV
48	0.495221	192.168.7.2	255.255.255.255	AODV
76	0.730169	192.168.7.2	255.255.255.255	AODV
97	0.939614	192.168.7.2	255.255.255.255	AODV
120	1.173552	192.168.7.2	255.255.255.255	AODV
143	1.414850	192.168.7.2	255.255.255.255	AODV
166	1.645481	192.168.7.2	255.255.255.255	AODV
187	1.852443	192.168.7.2	255.255.255.255	AODV
207	2.058690	192.168.7.2	255.255.255.255	AODV
229	2.277382	192.168.7.2	255.255.255.255	AODV
249	2.481370	192.168.7.2	255.255.255.255	AODV
271	2.707320	192.168.7.2	255.255.255.255	AODV
296	2.957814	192.168.7.2	255.255.255.255	AODV

Fig. H.8 Intervals de *hello* reals per HELLO_INTERVAL=100, cursant UDP

Podem veure com l'interval de *hello* a augmentat considerablement. En aquest cas, sí que transcorren més de 200 ms entre dos *hello* consecutius en més d'una ocasió, detectant, doncs, trencaments dels enllaços.

No entrarem en detall pels altres intervals de *hello*. En absència d'altre tràfic, o cursant TCP, la resta d'intervals són generats amb més exactitud com més grans són. En qualsevol cas, mai transcorre un temps major a ALLOWED_HELLO_LOSS * HELLO_INTERVAL entre dos *hello*s consecutius, sense haver-hi trencaments i, en conseqüència, tenint el ample de banda esperat. Com s'ha pogut observar dels resultats obtinguts a les proves amb

UDP, per a l'interval de 200 i 500 ms, també hem tingut problemes en els escenaris de 3 i 4 salts. En aquest cas, però, la complexitat és major, ja que existeixen més enllaços i més tràfic a la xarxa, empitjorant encara més les condicions anteriorment esmentades. A més, els talls han estat en menor quantitat i sense tenir conseqüències tant evidents en el ample de banda disponible.

H.5. Ample de banda disponible si l'interval de *hello* real fos el configurat

La prova següent ha estat per veure si realment les suposicions que hem fet en el apartat anterior són certes. Hem canviat el valor d'un altre paràmetre per defecte al RFC, en aquest cas, el ALLOWED_HELLO_LOSS. Així, allarguem el període de detecció de trencaments, i els enllaços no cauen degut al problema esmentat anteriorment. La corba d'ample de banda ha de ser l'esperada. Establim el valor de ALLOWED_HELLO_LOSS a 50. Hem de tenir en compte que aquesta modificació no té cap sentit en una xarxa real, almenys en una xarxa on els nodes es moguin amb certa freqüència, ja que no es detectarien els canvis en la topologia de la xarxa. A les següents figures mostrem els resultats d'ample de banda disponible introduint la nova variació.

A la figura H.9, la variació és mínima. En qualsevol cas, aquesta diferència també pot tenir origen en la alta inestabilitat del aire com a medi de transmissió. Ens hem trobat a vegades resultats que no eren coherents amb els anteriors si els nodes havien canviat de lloc. Les proves han estat comprovades una i altra vegada durant tot el treball per minimitzar aquests efectes en els resultats.

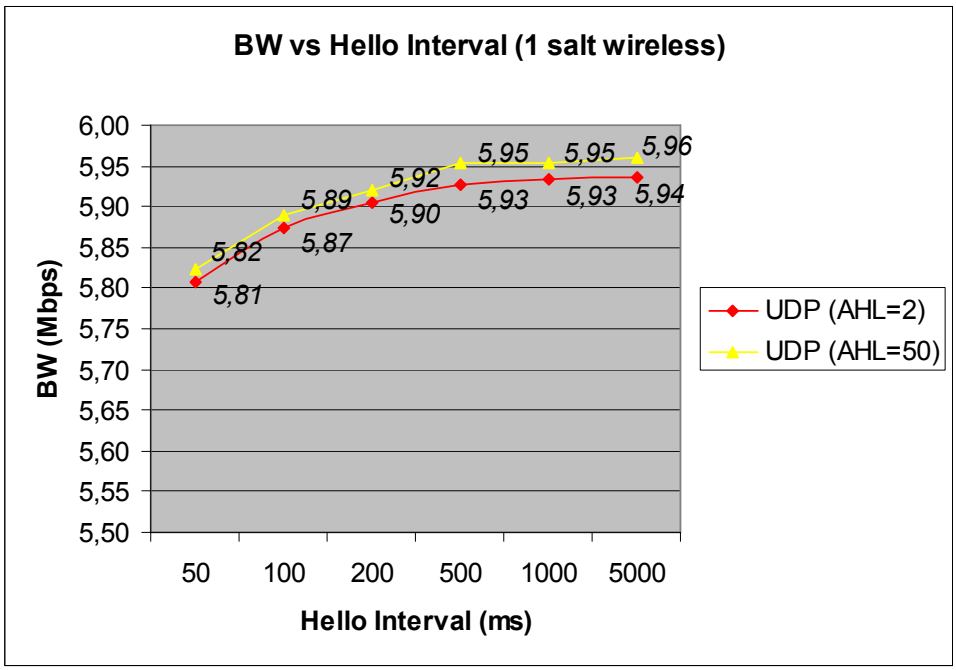


Fig. H.9 Variacions en el ample de banda en el escenari de 1 salt

A la següent figura mostrem els resultats per a l'escenari de dos salts. Passa exactament el mateix que a l'escenari anterior.

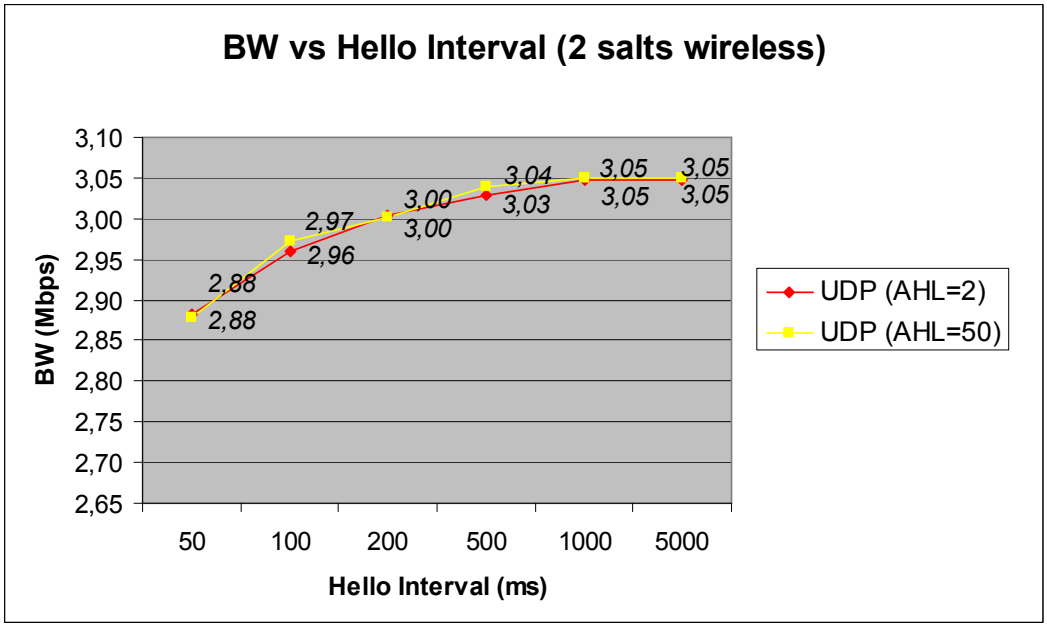


Fig. H.10 Variacions en el ample de banda en el escenari de 2 salts

La diferència entre les dues configuracions s'observa en els escenaris de 3 i 4 salts. Quan existeixen els trencaments, l'ample de banda disponible és molt menor, degut a la taxa de sortida real del missatges de *hello*. Amb la modificació del paràmetre ALLOWED_HELLO_LOSS, observem un comportament molt més lineal, disposant de major ample de banda per a l'aplicació, degut a l'absència de trencaments

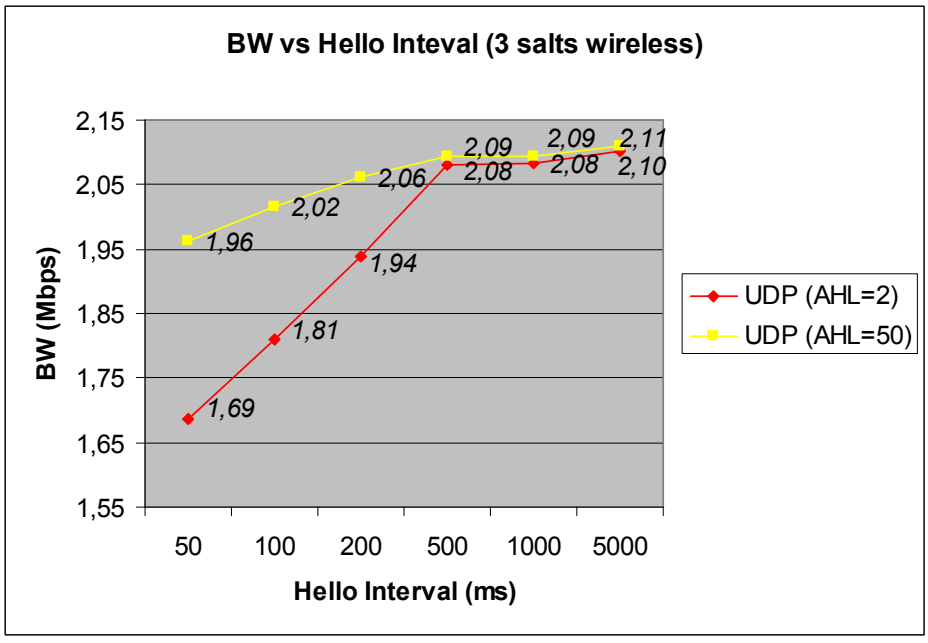


Fig. H.11 Variacions en el ample de banda en el escenari de 2 salts

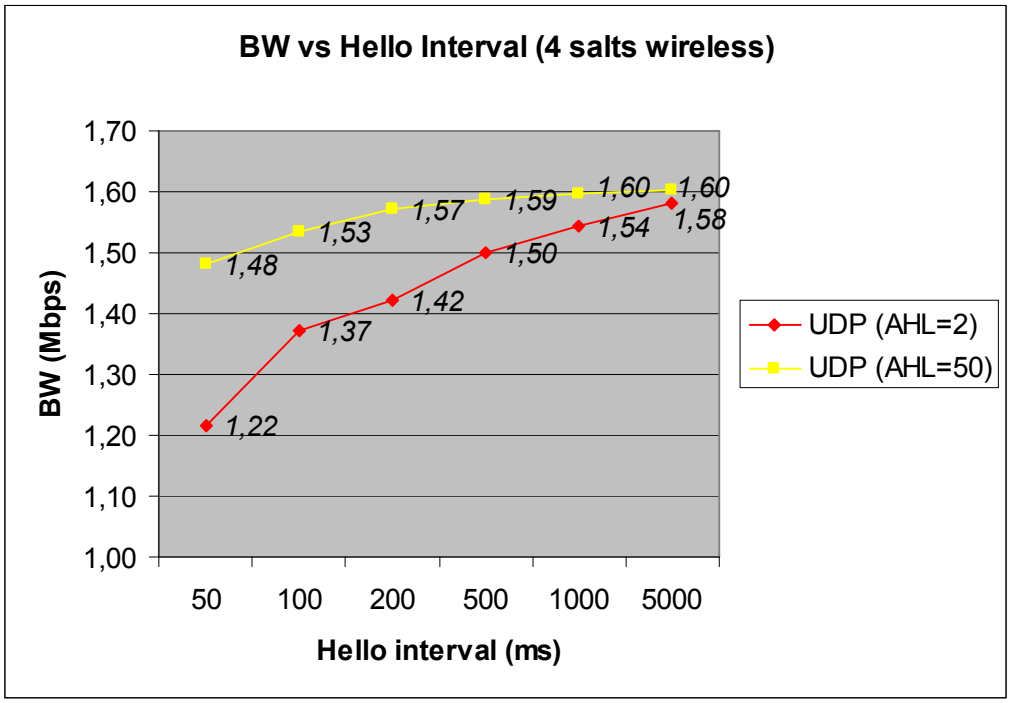


Fig. H.12 Variacions en el ample de banda en el escenari de 2 salts

H.6. Ampliació de l'anàlisi dels temps de *gap* obtinguts per TCP

En aquest apartat, justificarem els resultats obtinguts en la última part del treball, els relatius als temps de *gap* obtinguts amb TCP. Ja hem explicat com funcionen les retransmissions de TCP, i què hi tenen a veure en els temps de *gap*. El primer que farem es estudiar aquestes retransmissions, tant pel TCP de Linux com pel de FreeBSD. La primera d'aquestes retransmissions es calcula de manera diferent a Linux i FreeBSD. Les següents retransmissions es calculen segons el *back-off* ja mencionat anteriorment en tots dos casos. Analitzarem primer les retransmissions a Linux. La primera retransmissió es calcula segons el RTT obtingut al llarg de la transmissió TCP. Això es pot demostrar veient quines són les retransmissions fent la mateixa prova a Ethernet, ja que el RTT en aquestes xarxes és molt menor, degut a la major velocitat que disposem en aquest medi. Un cop establerta la comunicació entre origen i destí, hem desconnectat el node destí al cap de 10 segons. Esperem aquest temps perquè TCP hagi calculat un valor del RTT el prou precís. El valor de HELLO_INTERVAL l'establim a 5000 ms, ja que així podem observar més d'aquestes retransmissions. Capturant amb *Ethereal* a l'origen, podem veure quines són aquestes retransmissions i quin temps hi ha entre elles. De les proves realitzades, aquest són els valors que hem obtingut.

Taula H.7 RTOs del TCP Linux sobre 802.11b i Ethernet

Medi	1er RTO (s)	2on RTO (s)	3er RTO (s)	4rt RTO (s)	5è RTO (s)
802.11b	0,41	0,82	1,65	3,29	6,53
Ethernet	0,21	0,42	0,84	1,66	3,33

No.	Time	Source	Destination	rotocol	Info
3071	9.757858	192.168.7.5	192.168.7.2	TCP	ttcp > 32864 [ACK] Seq=1 Ack=2902825 Win=66608 Len=0
3072	9.757979	192.168.7.2	192.168.7.5	TCP	32864 > ttcp [ACK] Seq=2966537 Ack=1 Win=5840 Len=14
3073	9.758593	192.168.7.2	192.168.7.5	TCP	[TCP Window Full] 32864 > ttcp [ACK] Seq=2967985 Ack
3074	10.053855	192.168.7.5	255.255.255.2	AODV	Route Reply, D: 192.168.7.5, O: 192.168.7.5 Hcnt=0 D
3075	10.166239	192.168.7.2	192.168.7.5	TCP	[TCP Retransmission] 32864 > ttcp [ACK] Seq=2902825
3076	10.528187	192.168.7.2	255.255.255.2	AODV	Route Reply, D: 192.168.7.2, O: 192.168.7.2 Hcnt=0 D
3077	10.984166	192.168.7.2	192.168.7.5	TCP	[TCP Retransmission] 32864 > ttcp [ACK] Seq=2902825
3078	12.619905	192.168.7.2	192.168.7.5	TCP	[TCP Retransmission] 32864 > ttcp [ACK] Seq=2902825
3079	15.545429	192.168.7.2	255.255.255.2	AODV	Route Reply, D: 192.168.7.2, O: 192.168.7.2 Hcnt=0 D
3080	15.891422	192.168.7.2	192.168.7.5	TCP	[TCP Retransmission] 32864 > ttcp [ACK] Seq=2902825
3081	20.105574	192.168.7.4	192.168.7.2	AODV	Route Error, Dest Count=1
3082	22.434647	192.168.7.2	255.255.255.2	AODV	Route Request, D: 192.168.7.5, O: 192.168.7.2 Id=22
3083	22.439057	192.168.7.4	255.255.255.2	AODV	Route Request, D: 192.168.7.5, O: 192.168.7.2 Id=22
3084	22.442319	192.168.7.5	255.255.255.2	AODV	Route Reply, D: 192.168.7.5, O: 192.168.7.5 Hcnt=0 D
3085	22.914534	192.168.7.2	255.255.255.2	AODV	Route Request, D: 192.168.7.5, O: 192.168.7.2 Id=23
3086	22.917820	192.168.7.4	192.168.7.2	AODV	Route Reply, D: 192.168.7.5, O: 192.168.7.2 Hcnt=1 D
3087	22.918545	192.168.7.2	192.168.7.5	TCP	[TCP Retransmission] 32864 > ttcp [ACK] Seq=2902825
3088	22.932684	192.168.7.5	192.168.7.2	TCP	ttcp > 32864 [ACK] Seq=1 Ack=2936129 Win=66608 Len=0

Fig. H.13 Traça de les retransmissions de TCP Linux

En aquesta traça també podem veure el procés de descobriment de ruta, degut al trencament de l'enllaç. En realitat, el cinquè RTO expira just en el instant que s'envia el primer RREQ. Es vol transmetre el segment de TCP però no es pot perquè no tenim ruta, per tant, el següent no pot ser enviat fins que no termina el procés de descobriment. A la següent figura, mostrem la gràfica que segueix la evolució dels números de seqüència en funció dels temps, generada amb l'aplicació *Ethereal* (menú *Statistics -> TCP Stream Graph -> Time Sequence Graph)*. En aquesta figura podem veure els instants en els que TCP no transmet dades, i els instant en els que existeixen retransmissions. S'aprecia gràficament el *back-off* exponencial que segueixen aquestes retransmissions.

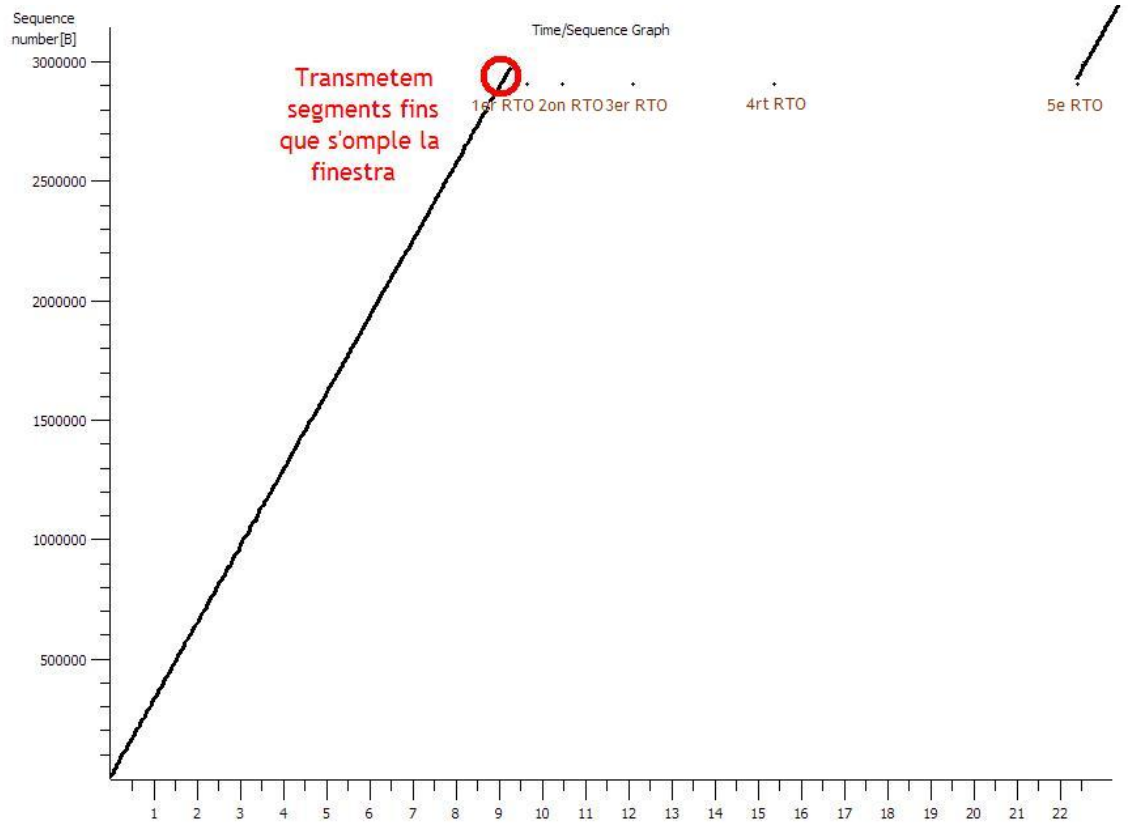


Fig. H.14 Números de seqüència durant el gap (TCP Linux)

Anem a veure ara quant valen aquestes retransmissions al TCP de FreeBSD.

Taula H.8 RTOs del TCP FreeBSD sobre el medi 802.11b

1er RTO (s)	2on RTO (s)	3er RTO (s)	4rt RTO (s)
1,02	2	4	8

No.	Time	Source	Destination	Protocol	Info
3319	11.588415	192.168.7.2	192.168.7.5	TCP	1032 > ttcp [ACK] Seq=3188665 Ack=1 Win=17520 Len=1
• • •					
3337	12.640303	192.168.7.2	192.168.7.5	TCP	[TCP Retransmission] 1032 > ttcp [ACK] Seq=3188665
• • •					
3340	14.640275	192.168.7.2	192.168.7.5	TCP	[TCP Retransmission] 1032 > ttcp [ACK] Seq=3188665
• • •					
3346	18.640173	192.168.7.2	192.168.7.5	TCP	[TCP Retransmission] 1032 > ttcp [ACK] Seq=3188665
• • •					
3361	26.640007	192.168.7.2	192.168.7.5	TCP	[TCP Retransmission] 1032 > ttcp [ACK] Seq=3188665

Fig. H.15 Traça de les retransmissions de TCP FreeBSD

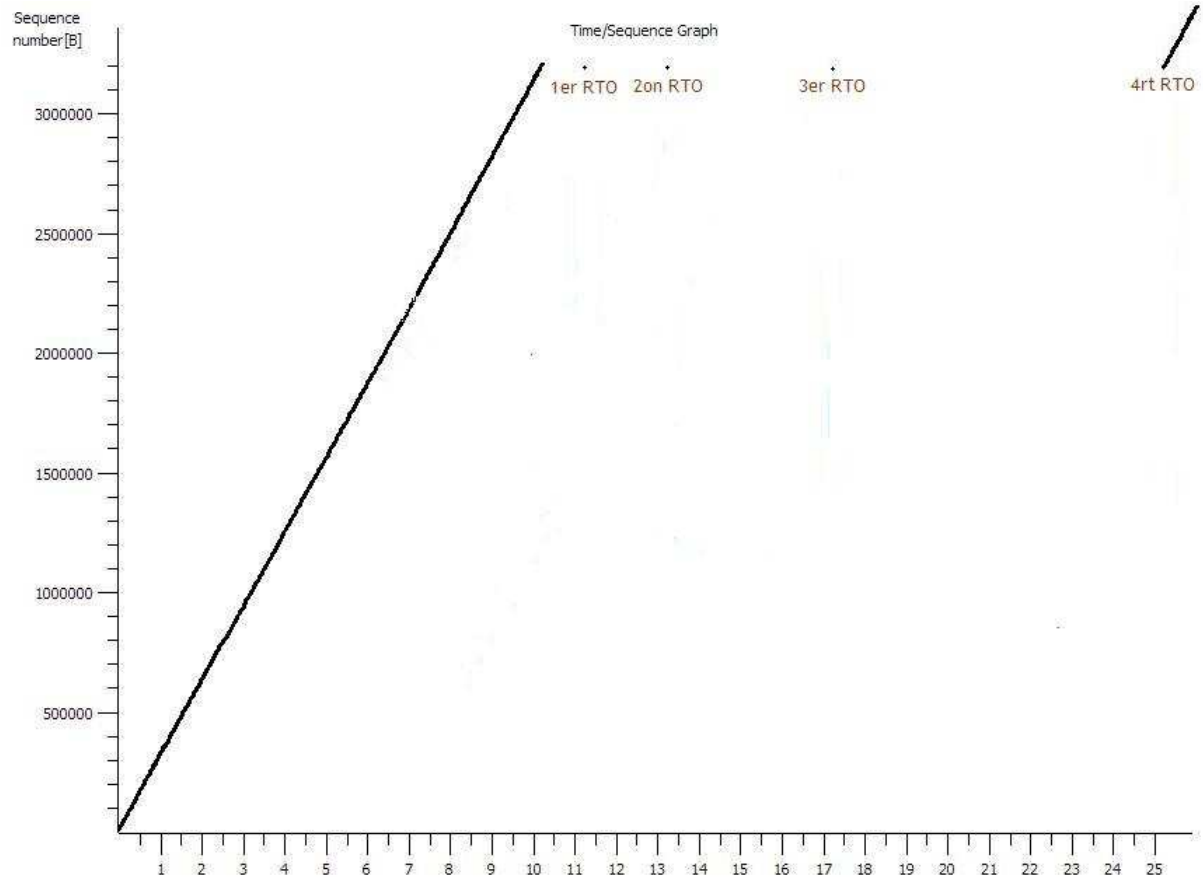


Fig. H.16 Números de seqüència durant el gap (TCP FreeBSD)

A FreeBSD, les retransmissions segueixen el *back-off,* però el primer RTO sempre expira al cap de 1 s. Això provoca que, per a un mateix valor de

l'interval de *hello*, els gaps de TCP FreeBSD siguin normalment majors als de Linux.

A continuació mostrem el diàleg TCP entre origen i destí en un d'aquests gaps, per entendre millor el funcionament.

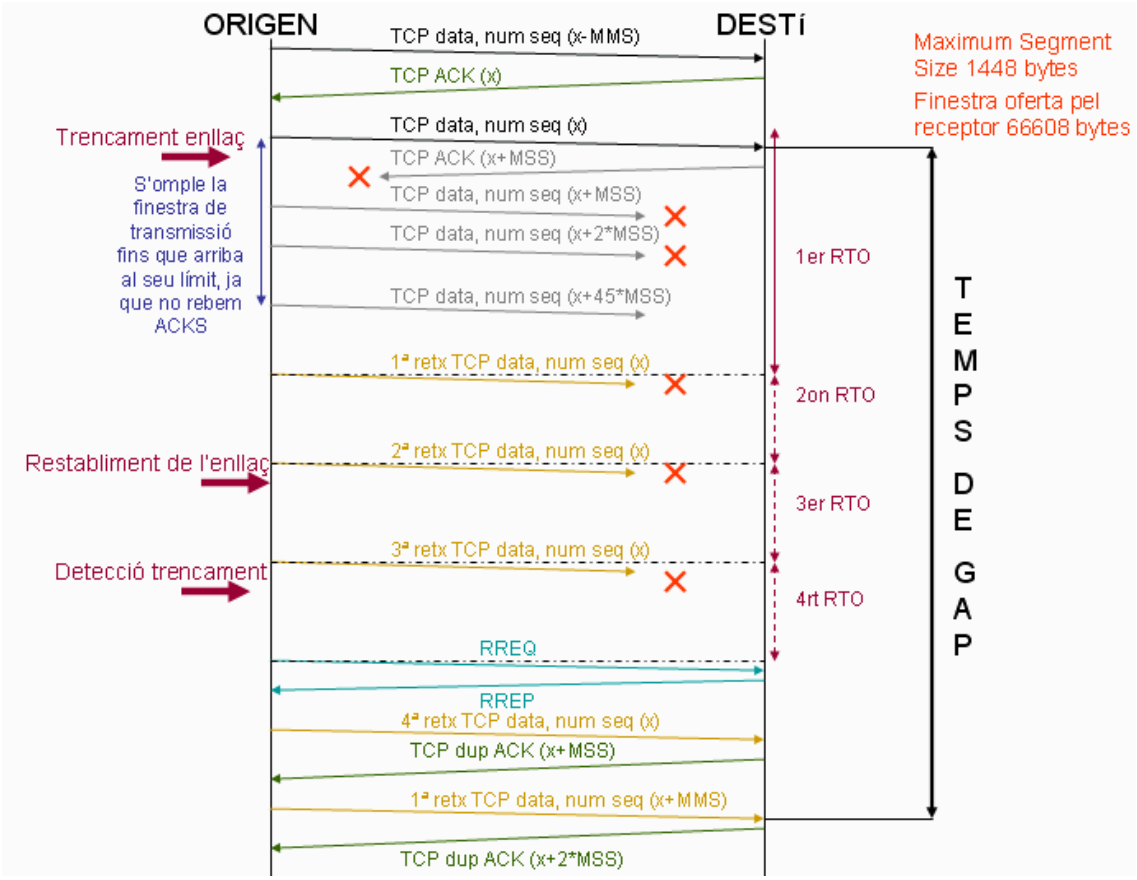


Fig. H.17 Diàleg TCP durant el trencament

La figura ajuda molt a entendre què és el que succeeix. En el moment que hi ha el trencament, ni les dades enviades per l'origen arriben al destí, ni els ACKs enviats en la direcció inversa arriben a l'origen. El origen pot enviar dades fins que s'ompli la finestra de transmissió oferta pel destí (66008 bytes). Per tant, el origen pot enviar fins a 46 segments fins que això succeeix.

Un cop assolit el valor de la finestra, el node no pot enviar més dades fins que no es reconeguin dades noves, i ha d'esperar la primera retransmissió del primer dels segments no reconeguts. Aquesta retransmissió no arriba al destí, ja que l'enllaç encara està trencat, per tant, seguim retransmetent. Les retransmissions no arriben al destí fins que AODV detecta el trencament i busca la nova ruta. És aleshores quan el destí reconeix dades noves (ja havia reconegut el segment, però l'ACK no havia arribat a l'origen per culpa del trencament), i la comunicació TCP segueix el seu camí.

A continuació anem a veure quant han durat aquest gaps en cadascun dels test, i en justificarem alguns dels resultats.

Taula H.9 Detall dels gaps obtinguts amb TCP Linux

Hello Interval (ms)	50	100	200	500	1000	2000	5000
Període de detecció (ms)	[50,100]	[100,200]	[200,400]	[500,1000]	[1000,2000]	[2000,4000]	[5000,10000]
Test #1	0,47	0,93	1,27	1,29	2,95	6,68	13,17
Test #2	0,46	0,47	1,27	1,26	2,89	6,1	6,24
Test #3	0,48	0,45	1,27	1,26	2,88	6,76	6,2
Test #4	0,47	0,46	1,24	1,27	2,91	6,1	6,2
Test #5	0,45	0,46	1,27	1,23	2,92	6,27	13,57
MITJA GAP UDP (s)	**0,47**	**0,55**	**1,26**	**1,26**	**2,91**	**6,38**	**9,08**

Taula H.10 Detall dels gaps obtinguts amb TCP FreeBSD

Hello Interval (ms)	50	100	200	500	1000	2000	5000
Període de detecció (ms)	[50,100]	[100,200]	[200,400]	[500,1000]	[1000,2000]	[2000,4000]	[5000,10000]
Test #1	1	0,99	3,01	2,98	3	6,96	15,02
Test #2	1,02	0,99	0,99	3,01	3,01	7,01	14,98
Test #3	1,04	1,01	2,96	3	3	7,01	15,02
Test #4	1	1,23	1,06	2,99	3,02	7,02	14,92
Test #5	1	1,06	1,06	2,98	2,98	7,02	15
MITJA GAP UDP (s)	**1,01**	**1,06**	**1,82**	**2,99**	**3,00**	**7,00**	**14,99**

Els resultats de color negre ja han quedat justificats amb les explicacions anteriors. Com veiem, el temps de *gap* és major que el temps de detecció, ja que hem d'esperar a la retransmissió de TCP, al nou descobriment i a la recepció de noves dades

Els resultats remarcats de color vermell són una mica més grans, degut a la misteriosa absorció del RREP pel node precursor al destí esmentada abans. En els resultats de color negre, això no ha succeït.

Els resultats de color blau ens indiquen un altre error de la implementació de AODV-UU, de la qual tampoc podem trobar explicació. Aquestes proves han estat realitzades amb els valors per defecte del RFC, per tant, el període de detecció és 2*HELLO_INTERVAL. En aquest casos, el protocol ha actuat de manera estranya, i la detecció s'ha dut a terme en 3 * HELLO_INTERVAL, tenint que esperar aleshores a la següent retransmissió de TCP. El més curiós és que la majoria de vegades que ha succeït això, ha estat utilitzant 200 ms per al valor d'interval de *hello*. Un altre cop, l'Erik Nordström no en coneixia la causa. És molt estrany, ja que en la resta de proves la detecció ha estat en el interval configurat.